Renato Paulo Simões da Silva Ribeiro

Licenciado em Ciências da Engenharia Eletrotécnica e de Computadores

Sistema de notificação para o setor de retalho baseado num motor de validação de regras da Internet das Coisas

Dissertação para obtenção do Grau de Mestre em

Engenharia Eletrotécnica e de Computadores

Orientador: Doutor Ricardo Luís Rosa Jardim, Professor Catedrático, Faculdade de Ciências e Tecnologia da Universidade Nova de Lisboa

Co-orientador: Doutor Carlos Manuel Melo Agostinho, Investiga- dor Integrado, Centro de Tecnologias e Sistemas, UNINOVA

Júri

Presidente:	Doutor José Manuel dos Santos Fonseca
Arguentes:	Doutor Carlos Eduardo Dias Coutinho
Vogais:	Doutor Carlos Manuel de Melo Agostinho

FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

Março 2019

Sistema de notificação para o sector do retalho baseado num motor de validação de regras da Internet das Coisas

AGRADECIMENTOS

Queria começar por deixar uns agradecimentos as pessoas que contribuíram para a realização desta dissertação.

Gostaria de agradecer ao meu orientador Professor Ricardo Gonçalves pela oportunidade dada. Agradeço também ao meu co-orientador Carlos Agostinho pelo tempo despendido ao longo da tese assim como ao David Aleixo da Knowledgebiz que esteve sempre pronto para tirar qualquer dúvida. Queria deixar um agradecimento geral a toda à gente da Knowledgebiz, fui muito bem recebido e especialmente à Ina Popescu que foi a minha colega de mesa durante algum tempo na empresa e ao João Bento e Nuno Carvalhão que participaram neste projeto comigo.

Deixo um grande agradecimento aos meus amigos com quem partilhei muitos bons momentos académicos durante a realização do curso Tomás Zúquete, Gonçalo Oliveira, Victor Fernandes, Diogo Proença, João Félix e Pedro Nunes.

Deixo também um agradecimento muito especial aos meus amigos mais próximos, nomeadamente, Elio Van-dúnem, Marcos Gonçalves, Dinaldo Rodrigues, Edson Nascimento e Tiago Mena Abrantes que contribuíram também para a realização deste trabalho indiretamente, quer seja, através de momentos de descontração ou de discursos de motivação.

Por fim queria deixar um agradecimento com muito carinho à minha família que sempre apoiou-me ao longo destes anos mesmo apesar da minha mudança de curso, em particular, à minha mãe Filomena Ribeiro, ao meu pai António Ribeiro, ao meu irmão Hugo, à minha irmã Elisandra e também à minha namorada Alexandra.

RESUMO

Com o aparecimento de novas tecnologias como o Big data, IoT e a evolução do *cloud computing* em novos paradigmas, a automatização de processos é cada vez mais uma necessidade. Esta automatização é encontrada em várias indústrias não só nas tecnologicas, mas na medicina, nos bancos, energias e retalho.

Com a análise de dados recolhidos pelos dispositivos utilizados hoje em dia quer sejam telemóveis, relógios ou mesmo os que estão em carros é possivel prever o comportamento humano, como atividades práticadas e a alimentação.

Esta tese vai-se focar no setor do retalho, onde todas essas características referidas anteriormente fazem a diferença. Pretende-se então informar as marcas acerca do comportamento do seu produto numa prateleira. O objetivo é diminuir os custos de recursos humanos e garantir a gestão eficiente do inventário. Esse objetivo vai ser concretizado por meio de tecnologias de processamento de eventos baseados em regras, como os sistemas de gestão de regras (BRMS). Estas tecnologias, dependentemente da regra associada executam uma acção, que neste caso vai ser uma notificação. A ferramenta BRMS, que contém um motor de regras para a gestão das regras, vai estar associada á um módulo de análise de dados para auxiliar na deteção de eventos que estão para acontecer.

Palavras-chave: Big Data, Internet das Coisas, Sistema de gestão de regras (BRMS), análise de dados, gestão de inventário

ABSTRACT

With the emergence of new technologies such as Big Data, IoT and the evolution of cloud computing into new paradigms, process automation is an increasing subject. This automation is found in various industries, not only in the technology, but also in medicine, banking, energy and retail.

Throught devices used today like smartphones, watches, or even cars it's possible to predict humam behavior such as physical activities or even eating habits.

This thesis will be focused on the retail sector, where those caracteristics play a important role. It's intended to inform producers of the behavior of their product on shelves. The objective is to decrease human resources and ensure de an efficient stock management. This objective will be fulfilled through the use of rule-based event processing technologies such as Business Rules Management Systems (BRMS). Technologies like this apply an action given a certain rule, which in this case will be a notification. The BRMS tool, thas a rule engine to manage the rules, will be connected to a analytics module to help on to detect events that haven't happened.

Keywords: Big Data, Internet of things, event processing, Business Rules Management System (BRMS), data analytics, stock management

ÍNDICE GERAL

LISTA DE TABELAS

LISTA DE FIGURAS

LISTA DE ABREVIATURAS

BRMS Business Rule Management System

CEP Complex Event Processing

EDA Event-Driven Architecture

HTTP Hypertext Transfer Protocol

IoT Internet of Things

JSON JavaScript Object Notation

MQTT Message Queuing Telemetry Transport

RFID Radio Frequency Identification

REST REpresentational Sate Transfer

SOA Service-Oriented Architecture

SOAP Simple Object Access protocol

TCP Transmission control protocol

XML Extensible Markup Language

1 INTRODUÇÃO

1.1 Visão do problema

O sector do retalho sofreu uma grande evolução ao longo dos anos, principalmente através do aumento do comércio online. Segundo a Ecommerce Association, o comércio online em Portugal aumentou cerca de 12,5% em 2017, sendo que o preço e a acessibilidade fácil são os principais motivos [1]. Apesar disso as lojas físicas são ainda importantes pois, constituem uma forma de manter uma ligação com o consumidor, sendo que o grande objetivo é garantir que este tem uma boa experiência.

Um dos aspetos fundamentais para o bom funcionamento de uma organização neste sector é a gestão do inventário. Uma boa gestão de inventário confere sempre uma vantagem em relação a competição. Um dos passos para permitir essa boa gestão, é a recolha de dados fidedignos, pois, através da informação retirada desses dados vai ser possível melhoria a nível operacional, de estratégia e produção.

Atualmente, esse processo de recolha de dados relativamente ao inventário e localização dos produtos é ainda feito de uma forma ineficiente, muitas vezes presencial. Com o aparecimento do conceito de Internet das Coisas (*IoT*) têm sido propostas cada vez mais soluções para este problema. Segundo Marjani M. *et al* [2], o *IoT* "oferece uma plataforma para que sensores e dispositivos se comuniquem perfeitamente em um ambiente inteligente e permite a partilha de informações entre plataformas de maneira conveniente."

As soluções desenvolvidas nesta área e para este tipo de problema envolvem a tecnologia Radio Frequency Identification (RFID). RFID é uma tecnologia que permite a comunicação através de etiquetas, colocadas em objetos, que contêm antenas [3]. Apesar da facilidade em instalar e apresentar maior segurança em relação aos códigos de barra, têm algumas desvantagens como:

- Custo de produção das etiquetas;

- Custo de mão-de-obra da colocação das etiquetas;
- Dificuldade na sua leitura em alguns objetos contendo líquidos ou feitos de metal;
- Tecnologia invasiva devido á capacidade de rastreamento.

A outra parte do problema passa não só pela deteção dos produtos, mas pela sua monitorização e notificação em tempo real. Na maior parte das vezes o comerciante não consegue controlar o estado do seu produto, isto é, quantas vezes foi movimentado e não foi comprado ou relocado para outra prateleira. Sendo que esta deteção de eventos numa prateleira é ainda hoje realizada manualmente, por componente humana. A falta de automatização pode levar a erros e a gestão de inventário menos eficientes.

1.2 Objetivo

Após a deliberação dos problemas apresentados é o objetivo da presente tese criar uma ferramenta de notificação e monitorização de eventos numa prateleira. Eventos como a quantidade de produtos, a localização e o estado devem ser controlados. Deverá existir um componente de análise de dados. Esse componente terá o objetivo de através dos dados acerca das vendas dos produtos retirar informação útil, como a previsão do comportamento do consumidor.

A presente dissertação foi feita em colaboração com a empresa KnowledgeBiz[1] e está enquadrada num projeto industrial. Portanto, há alguns requisitos que vão ter de ser respeitados que são:

- Implementação na *cloud*;
- Garantir a Escalabilidade;
- Comunicação entre vários componentes, ou seja, interoperabilidade;
- Interface acessível a pessoal menos técnico;
- Tecnologias *open source*;
- Custo de CPU e memória baixo.

1.3 Pergunta de investigação

A presente dissertação pretende responder à seguinte pergunta:

[1] http://www.knowledgebiz.pt

"Será possível, através da implementação de um sistema de notificação em tempo real, informar o utilizador acerca de eventos significativos que tenham ocorrido ou venham a ocorrer em expositores de produtos do sector do retalho?"

1.4 Hipótese

Com base na pergunta de investigação descrita na secção anterior e uma análise do estado de arte, no Capítulo 2, a hipótese proposta é a seguinte:

"Com a utilização de tecnologias de processamento de eventos baseadas em regras e com o apoio de um módulo de análise de dados, é possível detetar e prever eventos importantes em relação a produtos expostos no sector de retalho, permitindo assim uma rápida reação dos utilizadores responsáveis"

1.5 Organização do Documento

A presente dissertação é constituída por 6 capítulos que visam providenciar uma explicação detalha acerca do projeto desenvolvido. Esses capítulos são:

Capítulo 1: Introdução onde é realizada a contextualização da dissertação, abordando o problema, o objetivo, a pergunta de investigação e a hipótese.

Capítulo 2: Estado de Arte são apresentados fundamentos teóricos relacionados com o tema da dissertação assim como as suas implicações e potencial. Por fim, retiram-se conclusões gerais dos temas de maior importância.

Capítulo 3: Conceito Geral do Projeto foca-se no funcionamento do projeto num todo, assim como a apresentação da arquitetura geral e os conceitos mais determinantes.

Capítulo 4: Implementação é descrito mais profundamente o desenvolvimento do sistema assim como o seu funcionamento.

Capítulo 5: Testes e Validação de Resultados é focado nos ensaios realizados e os seus resultados.

Capítulo 6: Conclusões e Trabalho Futuro apresenta as impressões do sistema assim como melhorias e ideias que possam ser aplicadas futuramente.

2 ESTADO DE ARTE

O presente capítulo designa-se a abordar todos os temas relacionados com a implementação de um sistema para automatizar processos de monitorização de produtos em prateleiras. Primeiramente, são abordados os temas mais gerais como os paradigmas e as arquiteturas e no final a análise de dados. Entre estes dois temas serão descritos também alguns sistemas de processamento de eventos já existentes e as adjancentes ferramentas. Após todo este estudo são retiradas conclusões refletivas acerca dos referidos.

2.1 Paradigmas de computação em IoT

2.1.1 Cloud Computing

O conceito de *cloud computing,* permite a partilha de recursos e serviços através da internet [4]. Com o aumento do número de dispositivos a comunicarem entre si devido ao IoT, serão gerados cada vez mais dados, daí a integração do *cloud computing* nos sistemas IoT.

Apesar de serem mundos diferentes o *cloud computin*g e o IoT complementam-se visto que, o *cloud computing* ajuda o IoT a lidar com as suas deficiências tecnologias como o armazenamento, o processamento e a comunicação através dos seus recursos virtuais ilimitados. O *cloud computing* também beneficia da sua integração no IoT, pois aumenta o seu alcance para lidar com cenários do mundo real de maneira mais distribuída e dinâmica aumentando o número de serviços fornecidos[5].

2.1.2 *Fog e Edge Computing*

Com a evolução destes dispositivos IoT e a sua heterogenia, vai ser exigido mais da arquitetura *cloud*, pois, os dispositivos vão comunicar entre si cada vez mais e com maior frequência e variedade. Isto apresenta um problema para a *cloud*, porque enviar grandes quantidades de dados terá um elevado preço em aplicações que exijam menor latência e um custo mais alto de banda larga. Estes desafios potenciaram o aparecimento do *fog computing* [6].

O conceito de *fog computing* não tem o objetivo de substituir a *cloud*, mas sim complementar. Neste paradigma os nós *fog* vão estar situados entre a *cloud* e os dispositivos, com o objetivo de minimizar uma grande parte da comunicação feita através da internet, não sendo necessário o acesso à *cloud* [7]. Estes nós estão inseridos nesta nova camada virtual *fog* na rede de área local, como pode ser observado na Figura 2.1.

Após a revelação do conceito *fog* e o aperfeiçoamento dos dispositivos, surgiu o *edge computing*. Este conceito é aplicado numa camada ainda mais abaixo da camada *fog*, ou seja, na origem dos dados. Neste paradigma *edge*, os dispositivos que estão na origem dos dados vão ser responsáveis por várias tarefas como o processamento, tratamento e armazenamento de dados. Esta implementação traz melhorias à nível da redução de banda larga, consumo de energia e mesmo a nível de latência[8][9]. Na Figura 2.1 pode ser observada a camada *edge* localizada perto dos sensores e controladores.

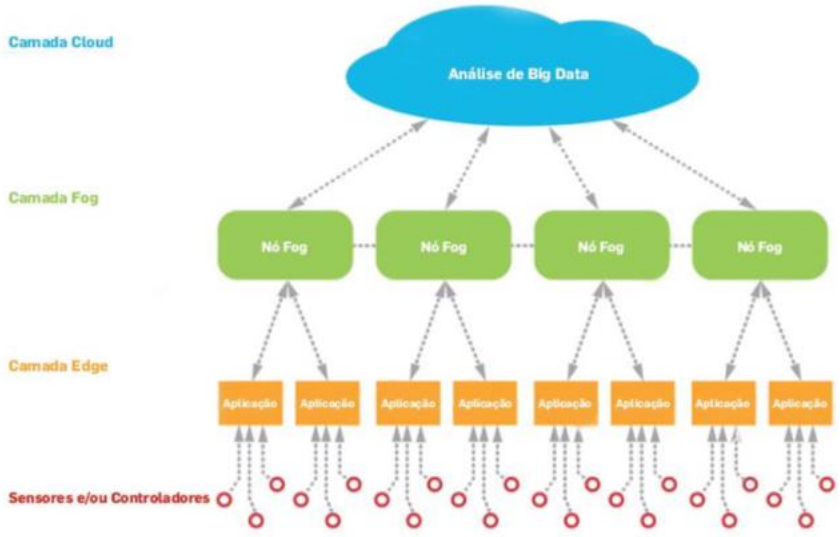


Figura 2.1 As camadas de *cloud*, *fog* e *edge computing* com os dispositivos físicos

2.2 Arquiteturas do IoT

2.2.1 Arquitetura Orientada a Serviços (SOA)

A Arquitetura Orientada a Serviço (*Service Oriented Architecture* - SOA) é um modelo para o desenvolvimento de aplicações distribuídas. Um serviço web é "uma coleção de operações que podem ser utilizadas através da internet por um cliente"[10]. Estes serviços são anunciados ou publicados por servidores para posteriormente serem cosumidos, como pode ser observado na Figura 2.2. A grande vantagem de uma arquitetura orientada a serviços é que permite reutilização de serviços, sendo que os serviços podem ser utilizados por outras aplicações. No entanto a SOA é mais dedicada à informação estática não lidando bem com informação dinâmica e aleatória [11].

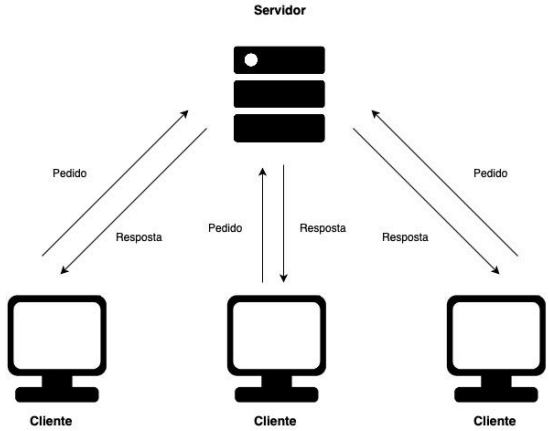


Figura 2.2 Modelo cliente-servidor

O protocolo responsável pela implementação de serviços web é o SOAP. O SOAP é um protocolo em que as mensagens de pedido e resposta estão formatadas no esquema XML para representar o conteúdo. Uma alternativa a este protocolo é a abordagem por REST (REpresentational State Transfer). Este estilo de arquitura permite a troca de mensagens em múltiplos formatos, como por exemplo por XML, mas também por JSON através de HTTP ou HTTPS.

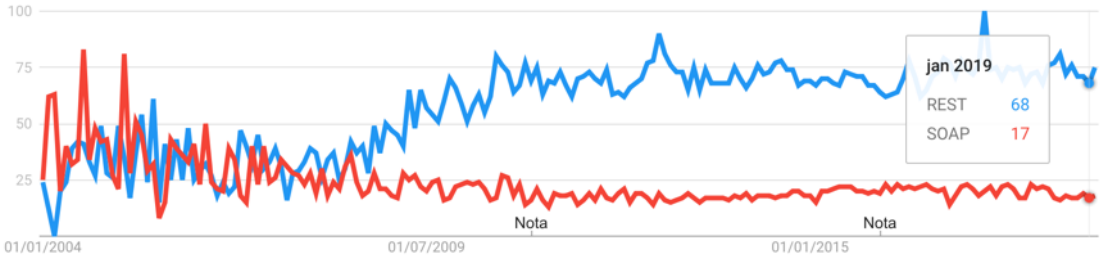


Figura 2.3 Comparação entre a abordagem REST (azul) e o protocolo SOAP (vermelho)

Apesar do protocolo SOAP ser ainda relevante nos dias de hoje, a abordagem por REST é mais utilizada devido a sua maior flexibilidade, escalabilidade e menor complexidade na implementação. Este fato pode ser observado na Figura 2.3 , um gráfico retirado do Google *trends* revelador da evolução da utilização dos dois.

2.2.2 Arquitetura Orientada a Eventos (EDA)

A Arquitetura Orientada a Eventos (*Event Driven Architecture* - EDA) é um sistema que permite que eventos sejam transmitidos entre componentes e serviços. Um evento é sempre que ocorre uma mudança de estado de um sistema, tanto pode ser uma interação no mundo real, como a deteção de uma presença ou no mundo virtual como a receção de uma mensagem.Sempre que há o desencadeamento de um evento este pode levar a outro ou ao pedido de um serviço [12].

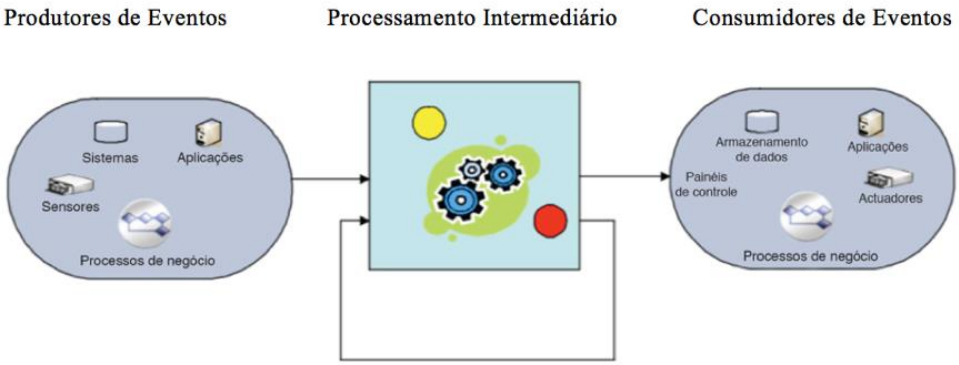


Figura 2.4 Exemplo de uma arquitetura orientada a eventos

Em comparação ao SOA, o EDA possui algumas diferenças, pois enquanto o SOA é desencadeado pelo cliente o EDA é desencadeado por eventos. EDA permite também uma comunicação entre "muitos-para-muitos" e assíncrona enquanto no SOA só existe comunicação síncrona e de "um –para-um"[13] [14].

Lina L. et. al. [14], defende que em uma plataforma IoT eficiente deve existir uma mistura das duas arquiteturas, tirando-se partido das vantagens das duas.

2.2.3 Padrão MVC

O padrão MVC é um padrão de arquitetura de *software* muito utilizado no desenvolvimento de aplicações web. Este padrão é constituído pelo modelo, o controlador e a visão, como pode ser observado na Figura 2.5. O funcionamento de cada um dos componentes é o seguinte:

Modelo – Situado no nível mais baixo, e é repsonsável por gerir e manter os dados.

Visão – A Visão é responsável pela apresentação de dados ao utilziador, sendo ativada sempre que o Controlador assim o decidir.

Controlador – O Controlador tem como funcionalidade garantir a interação entre o Modelo e a Visão, respondendo a entrada de dados feita pelo utilizador. Validando esses dados e depois a execução das operações de lógicas.

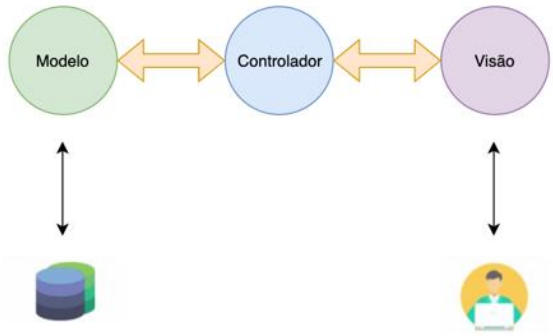


Figura 2.5 Funcionamento do padrão MVC

A grande vantagem do padrão MVC é a separação de tarefas. O Controlador recebe todos os pedidos do utilizador, que de seguida interage com o Modelo para realizar o tratamento dos dados. Esses dados são depois apresentados na Vista, como já tinha sido referido antes. O MVC é compatível com as arquiteturas anteriormente referidas, nomeadamente, no desenvolvimento de interfaces como é o caso da *framework* Angular.js[2].

[2] https://angularjs.org

2.3 Protocolos de comunicação em tempo real

Tendo em conta que algumas aplicações têm, como requisitos, alta interatividade e alterações de conteúdo visual imediato é necessário recorrer a tecnologias de comunicação em tempo real. As principais tecnologias são Websockets, AJAX e MQTT.

2.3.1 AJAX

Esta tecnologia funciona através de mensagens XML para o servidor. O AJAX tem várias formas de funcionamento *polling*, *long polling* e *streaming*. Na primeira forma o envio de pedidos é feito em intervalos de tempo regulares e assim que o servidor emite uma resposta é fechada a conexão. A segunda forma é muito parecida sendo que a única diferença é o que o intervalo e tempo é maior. Por *streaming* o servidor mantém a ligação aberta indefinitivamente até que seja o cliente a fechar.

2.3.2 Websockets

É um protocolo que permite comunicação totalmente bi-direcional (full duplex) e simultânea entre o cliente e o servidor. Este protocolo é assente no TCP e só utiliza o HTTP para operações de autorização e autenticação. Comparando com a tecnologia anterior, o AJAX requer um grande uso de memória e de banda larga da rede , devido aos longos processos de *looping* [15].

2.3.3 MQTT (Message Queing Telemetry Transport)

O protocolo MQTT é um protocolo de comunicação *publish/subscribe*, dos mais utilizados nos sistemas IoT. Este protocolo é baseado em uma arquitetura client/broker, em que o cliente publica dados no broker, que atua como mediador das mensagens. Os dados são publicados num certo tópico da mensagem e todos os outros clientes que estejam subscritos a esse tópico recebem essses dados. Os clientes podem estar subscritos a mais de um tópico. Uma das principais características do MQTT, são os seus três níveis de Quality of Service(QoS) garantindo a fiabilidade da entrega dos dados enviados [16].

Na Figura 2.6 pode-se observar a comunicação entre os *Publishers* e os *Subscribers*. Neste exemplo verifica-se que o *Subscriber5* está subscrito simultaneamente ao Tópico1 e ao Tópico2 recebendo assim mensagens do *Publisher1* e *Publisher*2.

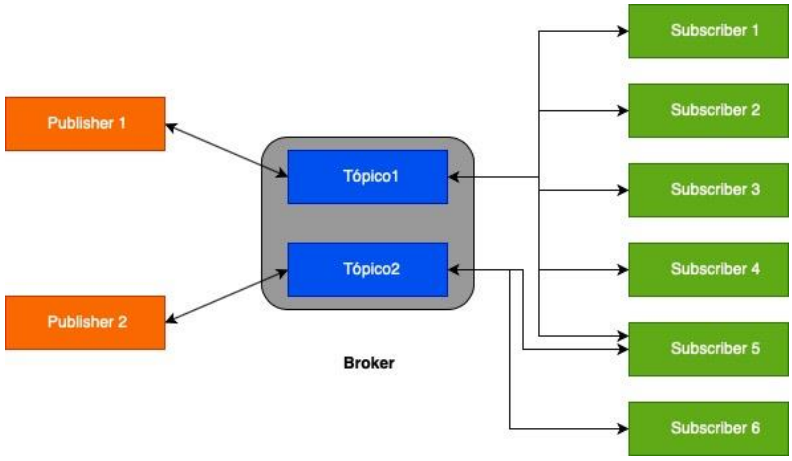


Figura 2.6 Funcionamento do protocolo MQTT

2.4 Processamento de eventos baseados em regras

O processamento de eventos baseado em regras é um conceito em que através de um fluxo de eventos é possível extrair informação útil, usando regras pré-definas pelo utilizador. Uma regra pode ser vista como uma instrução que é aplicada a determinada situação. Pode ser observado de seguida um exemplo teórico de uma regra.

Se

A 'Quantidade de Objetos' na prateleira é menor que 10

Então

O 'estado' da Prateleira é 'crítico'

Podem de seguida ser encontrados alguns exemplos em indústrias do processamento de eventos na Tabela 2.1:

Tabela 2.1 Exemplos da aplicação do processamento de eventos nas indústrias

Indústria	**Exemplo de aplicação**
Saúde	Monitorização do estado de saúde de bebés recém-nascidos [17].
Energia	Deteção de anomalias em *smart grid* [18].

Transportes	Monitorização das ruas mais visitadas de uma localização através de fluxos de data Geo-espacial [19].
Bancária	Deteção de fraude e transferências suspeitas [20].

2.4.1 Sistemas de Processamento de Eventos Complexos

O processamento de eventos complexos (CEP) é a combinação do processamento de eventos com um grande volume de dados provenientes de várias fontes. O modelo CEP depende da capacidade de especificar eventos complexos através do conteúdo da notificação de eventos recebidos pelos produtores de eventos. Esses eventos podem ter acontecido no mundo real. Esse conteúdo é filtrado e agregado com outras notificações, num evento composto, para ser possível obter conclusões e tomar decisões como, a notificação dos consumidores de eventos, como pode ser visualizado na Figura 2.7 [21].

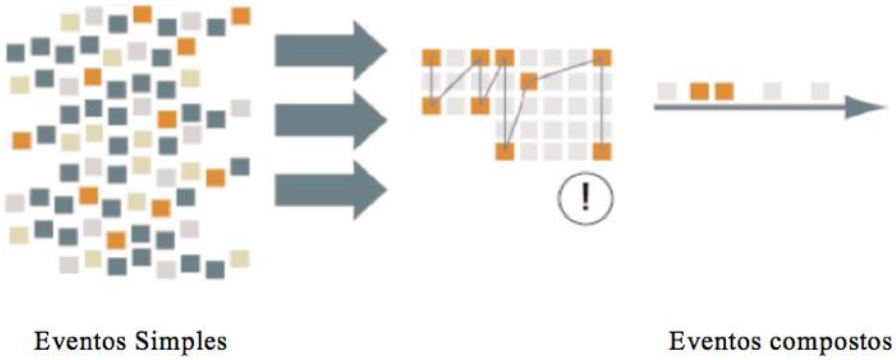


Figura 2.7 Processamento de eventos complexos

O objetivo de uma ferramenta CEP é identificar eventos significativos, como oportunidades ou ameaças, e responder a eles o mais rápido possível. Comparando com outros sistemas de processamento de eventos as ferramentas CEP apresentam algumas vantagens e desvantagens que são:

Vantagens[22][23]:

- Processamento de um grande volume de dados;
- Aquisição e tratamento de dados em tempo real;
- Planeamento de padrões para comportamentos de risco.

Desvantagens[22][23]:

- Dificuldade em lidar com fluxos de dados atrasados ou fora de ordem;

- Pode gerar falsos positivos.

2.4.1.1 Ferramentas CEP

As ferramentas CEP são denominadas processadores de eventos complexos. Existem já uma grande variedade de ferramentas disponíveis. Com a análise de ferramentas CEP deve-se ter em conta alguns requisitos, nomeadamente, escalabilidade, baixa latência e elevado processamento.

Escalabilidade: garantir que a ferramenta consiga manter o seu funcionamento com a variação do volume de dados, pois os eventos são assíncronos e a informação dinâmica consoante também a variação das fontes de dados.

Baixa latência: eventos têm de ser respondidos em tempo real, ou seja, com o mínimo atraso possível. Eventos importantes, como por exemplo fraude bancária, têm de ser respondidos rapidamente.

Elevado Processamento: estas ferramentas lidam com grandes quantidades de dados e têm de ser capazes de processar essa mesma quantidade de dados.

Após a descrição de alguns requisitos relevantes na escolha de uma ferramenta CEP, pode ser observada na Tabela 2.2 um estudo dessas ferramentas.

Tabela 2.2 Ferramentas CEP e as suas características

Ferramentas	Escalabilidade	Baixa Latência	Elevado Processamento	Open Source	Tipo de Documentação	Linguagem de Programação
WSO2 CEP	●	●	●	●	Documentação online e comunidade	Java
TIBCO BusinessEvent	●	●	●		Documentação online	Java
ESPER	●	●	●	●	Documentação online e comunidade	Java
Dynamic CEP(DyCEP)	●	●	●	●	Pouca Documentação	Java
SASE	●	●	●	●	Documentação online e comunidade	SQL
FUJITSU Interstage	●	●	●		Documentação online	Java

As ferramentas estudadas foram o WSO2 CEP[24], TIBCO BusinessEvents[25], ESPER[26], Dynamic CEP[27], SASE [28], FUJITSU Interstage[29].

- **WSO2 CEP**

O WSO2 CEP é uma ferramenta *open source* de processamento de eventos complexos baseada na linguagem de programação Java e com uma aproximação da linguagem SQL. Esta ferramenta utiliza a biblioteca Java WSO2 Siddhi[30] para um elevado processamento e o Apache Storm para a escalabilidade. A biblioteca WSO2 Siddhi é usada para deteção de eventos e desencadeamento de ações e o Apache Storm é um sistema distribuído para o processamento de Big Data [24].

- **TIBCO BusinessEvents**

A ferramenta TIBCO BusinessEvents é não só um processador de eventos complexos, mas também um motor de de regras em tempo real. Esta ferramenta respeita todas as características de um CEP, mas as regras são definidas como expressões declarativas como num motor de regras. É considerado um motor de regras por estado, pois o armazenamento de regras nunca esta em pausa assim os eventos futuros podem ser sempre comparados a eventos passados. Esta ferramenta não é *open source* [25].

- **ESPER**

A seguinte ferramenta estudada foi o Esper, esta ferramenta é baseada em Java, mas contém uma linguagem parecida com a SQL, denominada a língua de processamento de eventos (EPL). EPL é uma das grandes vantagens do ESPER, pois é utilizada para os fluxos de eventos e padrões[26].

- **CEP Dinâmico (DyCEP)**

O DyCEP é a nova geração de processadores de eventos complexos, que permite detetar e responder em tempo real, sem atraso ou mesmo antes do tempo. A essa descoberta de situações complexas denomina-se reatividade complexa. Uma das grandes vantagens do DyCEP é a existência de canais entre componentes de processamento criando uma rede de processamento complexo de eventos, como pode ser observado na Figura 2.8. O DyCEP não dispõe de muita documentação online nem comunidade [27].

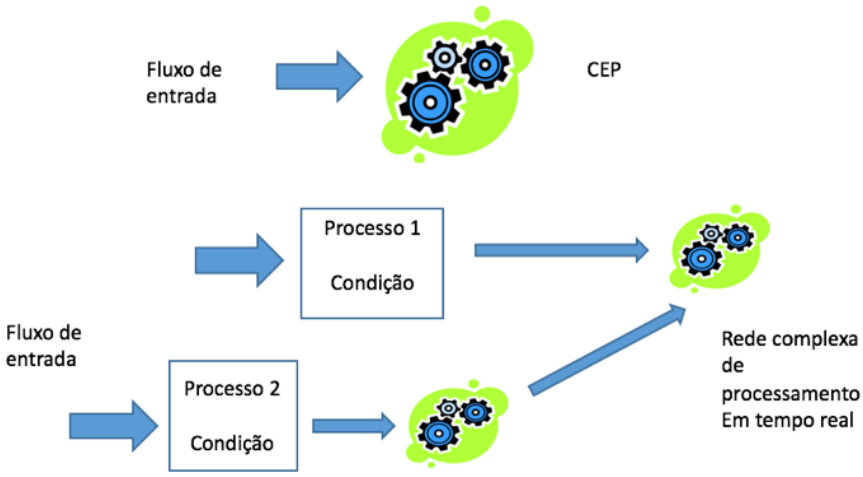


Figura 2.8 Comparação entre a arquitetura de um CEP e a do CEP Dinâmico [16]

- **SASE**

Á semelhança de algumas das outras ferramentas, a SASE baseia-se numa linguagem parecida com a SQL para diminuir complexidade. A arquitetura desta ferramenta é constituída por três camadas: a camada física do dispositivo, a camada de associação e limpeza e a camada do CEP.A camada física é onde se encontra o dispositivo e a de associação e limpeza é onde ocorrem os processos de filtragem e utilização de atributos, como a data para a criação de eventos e mais tarde ajudar na decisão. A última cama é o processador de eventos [28].

- **FUJISTSU Interstage**

O FUJISTSU Interstage não é uma ferramenta *open source* pois, é necessária uma licença comercial. Como algumas ferramentas anteriormente referidas baseia-se também numa linguagem parecida com o SQL para a atribuição de regras. A grande vantagem desta ferramenta é que funciona em paralelo com a tecnologia Hadoop, ferramenta responsável pelo processamento e análise de Big Data [29].

Dentro dos casos estudados as ferramentas mais interessantes para aplicação na presente dissertação seriam o ESPER e o WSO2 CEP. Estas ferramentas para além de serem *open source*, contêm documentação online e comunidade académica ativa. Apesar disso o WSO2 CEP é o que apresenta mais documentação.

2.4.2 Sistemas de Gestão de Regras de Negócio (BRMS)

O principal objetivo de um sistema de gestão de regras é automatizar decisões consoante as regras. As regras são expressões declarativas, com linguagem simples e acessível, relacionadas diretamente com os aspetos de um certo negócio. Essas regras são implementadas longe do código permitindo que a sua lógica possa ser utilizada por outras aplicações [31].

Todos os BRMS têm associado a si um motor de regras, que é simplesmente um interpretador "se-então". Hayes-Roth [32] , já descrevia os motores de regras há algumas décadas. Este componente tem capacidade de executar uma ação através dos dados que recebe. Essas ações são decididas pelo utilizador na criação das regras [33]. Um BRMS apresenta algumas vantagens e desvantagens em comparação a outros sistemas de processamento de eventos que são:

Vantagens [31]:

- A utilização de regras é mais fácil de interpretar do que código, ou seja, é mais acessível a pessoal não técnico.
- Facilidade em lidar com o aumento de complexidade, caso seja feita alguma mudança no sistema basta modificar ou adicionar uma nova regra.
- Capacidade de resolver problemas de maior dificuldade, informando qual foi a decisão tomada e o porquê.
- Automatização de processos levando a melhorias na eficiência e produtividade
- Flexibilidade com arquiteturas orientadas a serviço e a eventos

Desvantagens[31]:

- Não lida bem com variáveis que apresentam um número infinito de valores.
- Não aprende as regras depende da criação de novas regras.

2.4.2.1 Algoritmos para correspondência de regras

Nesta secção serão expostos os algoritmos mais utilizados nos motores de regras, servindo estes para os distinguir em termos de capacidades.

Forward Chaining: neste algoritmo é priorizado o facto, ou seja, pode haver correspondência de duas regras. Esse facto propaga-se e chegamos a uma conclusão ou ação, por isso é considerado um algoritmo orientado a dados.

Backward Chaining: este algoritmo é orientado a objetivos tendo em conta que é sabida uma conclusão, o sistema vai tentar satisfazer essa conclusão através de sub-objetivos. Este processo continua até a conclusão inicial ser satisfeita ou quando deixarem de existir sub-objetivos.

A diferença entre os dois algoritmos pode ser observada no exemplo apresentado na Figura 2.9.

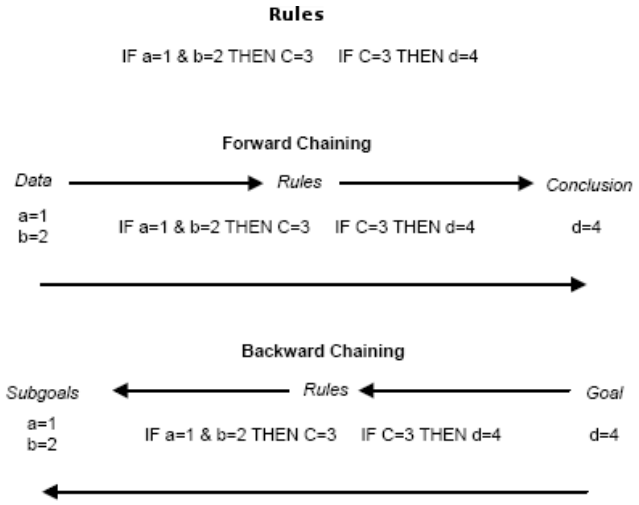


Figura 2.9 Comparação entre forward e backward chaining

2.4.2.2 Ferramentas BRMS

- **Drools**

O Drools é uma ferramenta *open source* escrita em java, que providencia uma plataforma integrada de regras e processamento de eventos. Esta plataforma é baseada no algoritmo *forward chaining* e no algoritmo de Rete para executar as regras de forma eficiente. O algoritmo de Rete tem o objetivo melhorar a velocidade das regras de encadeamento progressivo através do reconhecimento de padrões. Um a ação pode ser desencadeada ao existir correspondência de certas regras. Através do algoritmo de Rete não é necessário fazer as verificações dessas regras todas pois ele lembra-se dos testes anteriores, retirando a ineficiência de ir testar as regras outra vez. Este algoritmo tem a única desvantagem de um custo elevado de memória.

- **Notification Enabler (NE)**

O Notification Enabler é uma ferramenta BRMS baseada em linguagem de programação Javascript. Esta ferramenta é utilizada no projeto vf-OS[3](virtual factory Open Operating System), um projeto direcionado a implementação de soluções ICT na área da manufactura. Entre essas soluções encontram-se os *enablers*. O vf-OS engloba inúmeros enablers que têm várias funcionalidades como o processamento de eventos e fluxos, gestão de contexto e controlo da rede e da informação. O *notification enabler* providencia as aplicações e componentes vf-OS um sistema de notificações de simples utilização. Este sistema permite aos utilizadores a criação de regras e a gestão de notificações. Com a recepção de dados os mesmos são comparados com as regras. Existindo correspondência o utilizador é informado via mail ou via a aplicação.

[3] https://www.vf-os.eu

2.5 Análise de Dados

2.5.1 Big Data em IoT

Tendo em conta a evolução das principais indústrias, o *Big Data* vai ter um papel muito importante na indústria 4.0. A indústria 4.0 é a integração do IoT num contexto de fábricas [34].

O aumento do volume dos dados em simultâneo com a variedade de fontes de dados e velocidade a que estes dados são gerados, surgiram como as principais características do *Big Data*. Com a necessidade do estudo dos dados para descobrir padrões novos, revelar tendências, correlações desconhecidas e novas informações que possam beneficiar as organizações a tomar decisões eficientes e bem ponderadas foram definidos novos vs que são a veracidade e o valor[35].

Volume: devido a enorme quantidade de dados gerados o volume é um fator importante. Essa grande quantidade de dados terá de ser armazenada e processada de forma eficiente[36].

Variedade: refere-se aos vários tipos de dados, estruturados ou semiestruturados. Esta informação é necessária porque pode ser necessária haver pré-processamento retirando o mais importante, como por exemplo áudio e vídeo[36].

Velocidade: com o aumento desse volume de dados, a velocidade a que eles são gerados e transmitidos tem um peso enorme no seu processamento. Esse processamento em muitos dos casos terá de ser em tempo real porque a resposta não poderá sofrer um grande atraso[36].

Veracidade: refere-se á fiabilidade dos dados**,** se são ou não de confiança. O estudo através de dados não credíveis pode levar a conclusões erradas ou irrelevantes [35].

Valor: refere-se ao processo de obtenção de novos conhecimentos através da análise dos dados [36].

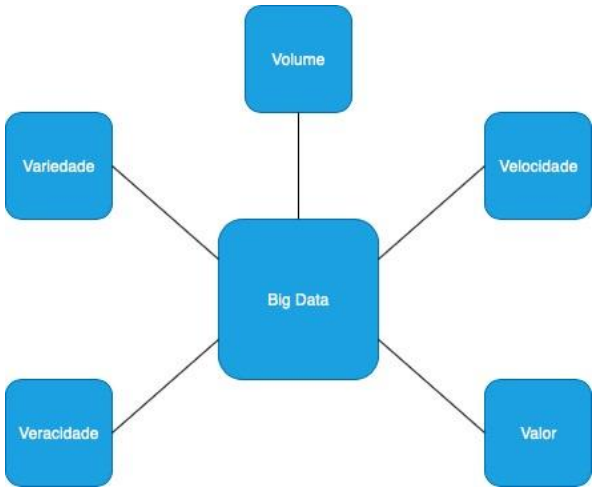


Figura 2.10 Os 5 v's do Big Data

2.5.2 Metodologias para a análise de dados

Dentro da análise de dados existem vários métodos, sendo os mais comuns a visualização dos dados, análise estatística e o *data mining* [36].

Visualização de dados: permite mudar a forma como se interage com a informação retirada dos dados é apresentada. Essa apresentação pode ser em forma de gráficos ou mapas, melhorando assim a compreensão e comunicação de dados.

Análise estatística: é a aplicação de conhecimentos da área da estatística, que é uma subárea da matemática. Através dos dados vai ser possível fazer análises descritivas e preditivas. As análises descritivas são utilizadas para resumir e descrever uma certa quantidade de dados e as preditivas para prever comportamentos futuro.

Data mining: consiste na procura de padrões e modelos que estão implícitos nos dados, com o objetivo de encontrar relações desconhecidas. Uma grande parte dos algoritmos de data mining foram desenvolvidos nas áreas de aprendizagem automática e inteligência artificial. Algoritmos como o Support Vector Machine(SVM), k –means e o naive bayes[37].

2.5.3 Relação entre IoT e Análise de *Big Data*

Num ambiente IoT, a análise de *Big Data* tem um grande potencial para extrair informação significativa de dados dos dispositivos. Para esse ambiente ter sucesso é requerido interoperabilidade, compatibilidade, confiabilidade e eficácia das operações a larga escala [38].

Interoperabilidade e Compatibilidade: comunicação entre dispositivos heterogéneos e utilização dos mesmos serviços.

Confiabilidade: obtenção de dados de qualidade com formatos consistentes.

Eficácia: garantir que são aplicados os processos e tecnologias corretas para obter resultados positivos.

Não só o volume dos dados como as outras características irão ultrapassar as atuais arquiteturas das empresas, como a análise de tempo real também terá uma grande influência e impacto nas capacidades de processamento da empresa. É importante entender que o sucesso da IoT está na incorporação da análise de *Big Data*, como pode ser observado na Figura 2.11. Podemos observar a camada dos dispositivos/sensores que estão conectados através de uma rede sem fios como por exemplos, RFID, WiFi ou Bluetooh. A gateway possibilita a comunicação entre a internet. Nas camadas acima temos o processamento de eventos e o armazenamento de dados, que estão interligados com a camada de análise de *Big Data*. As aplicações contêm assim uma gestão por API e um componente para a visualização dos dados.

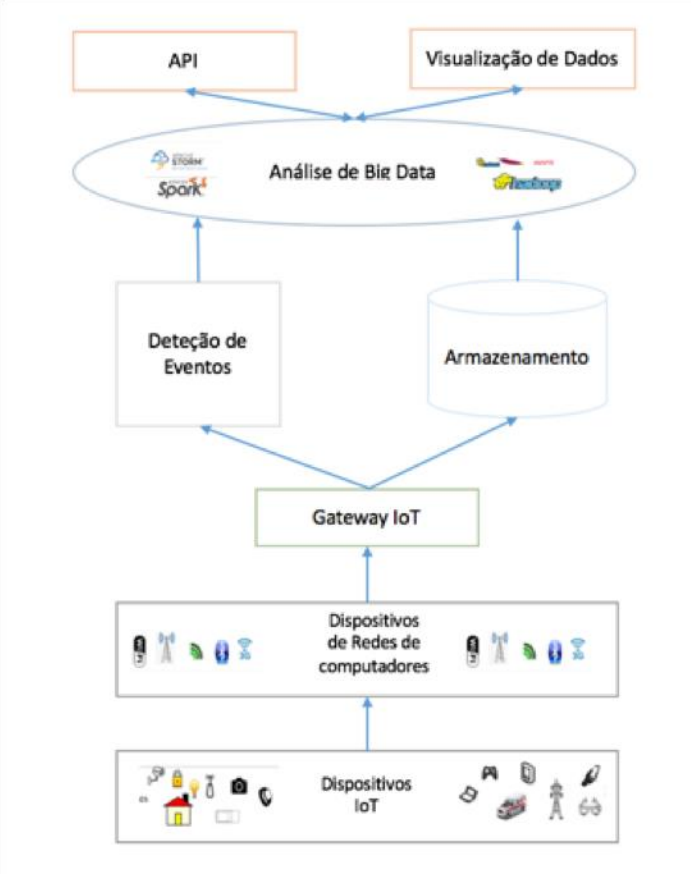


Figura 2.11 Arquitetura IoT com a análise de Big Data

2.5.4 Tecnologias de Análise de Dados

Embora os princípios e abordagens para a análise de dados sejam frequentemente discutidos, não existem ainda muitas tecnologias que sejam convenientes para lidar com as duas principais características, como o volume e a velocidade dos dados. Atualmente as tecnologias que lidam com essas características oferecem uma abordagem de computação distribuída para lidar com esses dois problemas [36]. Na presente secção seram discutidas algumas dessas ferramentas.

- **Hadoop**

O ecossistema Hadoop oferece uma solução escalável para carregar, armazenar e pré processar uma grande quantidade de dados. A tecnologia Hadoop é baseada no paradigma MapReduce, que consiste em fazer uma divisão de tarefas tornando as tarefas mais eficientes apresentando um só resultado na saída [39]. Esse paradigma está dividido em dois passos:

Map: vai realizando as operações a partes de dados apresentando resultados intermediários.

Reduce: através dos resultados intermediários, é feito o processamento e apresentado um resultado final.

A plataforma hadoop é ainda constituída por mais dois componentes, o Sistema de Dribuição de Ficheiros Hadoop (*Hadoop Distributed File system* - HDFS) e o Yet *Another Resource Negotiator* (YARN).

O HDFS é um sistema de ficheiros para o armazenamento e distribuição de dados por nós. Este sistema foi concebido para correr em *harware* de custo muito reduzido, pois, vários nós têm armazenados dados [40].

O YARN surge do problema da utilização de recursos através das *frameworks* de processamento no ecossistema Hadoop. Este é responsável pela alocação de recursos, que são dinamicamente utilizados pelas aplicações, permitindo combater problemas de sobreposição. Concluindo, as capacidades computacionais serão divididas de forma eficiente[41]. O Hadoop não é adequado para o processamento de arquivos pequenos. Embora existam algumas ferramentas adicionais ou abordagens do Hadoop que permitam o processamento de arquivos pequenos, elas vêm com muitas restrições.

- **Apache Spark**

É uma ferramenta de processamento de dados *open source*, para executar análise de dados em sistemas de computação distribuída, como o Hadoop. O Spark suporta o processamento na memória para aumentar a velocidade e o processo de dados sobre o MapReduce. O Spark é implementado na parte superior do *cluster* do Hadoop existente, o que permite que o Spark aceda aos dados por meio do HDFS [42]. Como uma alternativa ao tradicional modelo MapReduce, o Spark vem com um novo modelo de computação que é supostamente mais eficiente que o Hadoop em áreas como fluxos em tempo real e algoritmos iterativos comuns em aplicações gráficas e aprendizagem automática. Além disso, também é considerado um ambiente eficiente para ferramentas interativas de data mining[42].

- **Linguagem de programação R e Python**

R, é uma linguagem de programação estatística e gráfica que é agora aplicada na área de análise de dados. Uma das grandes vantagens desta tecnologia é ser *open source* e tem uma grande comunidade. Tem uma grande variedade de pacotes e é uma linguagem flexível. Tal como a linguagem R, o Python é também uma linguagem usada na análise de dados com a vantagem de ser *open source*. Contém já algumas ferramentas preparadas para lidar com a análise de dados como o Panda, Numpy, Scipy. Esta linguagem contém várias bibliotecas criadas úteis para a análise de dados e é facilmente integrável com outras aplicações.

2.6 Conclusões Gerais

Através do estudo realizado no presente capítulo foi possível abordar as várias tecnologias e ferramentas com o fim de implementar e desenvolver o sistema proposto e responder à pergunta de investigação. Esse sistema será um sistema inteligente para a monitorização de prateleiras em um ambiente *cloud* baseado no conceito IoT. Apesar desse estudo é necessário ter também em conta os requisitos enunciados na secção 1.2.

Sendo o sistema proposto baseado no conceito de IoT, esse terá de ser flexível para fazer a ponte entre o mundo físico e o das tecnologias de informação. Para garantir esta flexibilidade o sistema deveria ser implementado com a combinação entre o paradigma SOA e EDA pois, cada evento que ocorra no mundo físico terá de desencadear um serviço.

Adicionalmente, um dos grandes objetivos do sistema é a tomada de decisões por parte dos utilizadores. A eficiência dessas decisões depende da velocidade a que a informação é apresentada, como tal, a comunicação em tempo real será realizada por intermédio dos protocolos de comunicação MQTT e Websockets.

Para o processamento de eventos os sistemas CEP são uma tecnologia muito interessante, como foi demonstrado na secção 2.4.1. As ferramentas ESPER e WSO2 CEP têm licença open source e são normalmente utilizadas em ambiente académico devido á facilidade de obter apoio técnico na internet. Sendo que um dos objetivos é a facilidade da introdução de regras por parte de pessoal menos técnico a opção mais óbvia seria um sistema de gestão de regras(BRMS). Esses sistemas são conhecidos pela sua simples implementação de regras e baixo custo computacional. Têm também a capacidade de serem facilmente integráveis em qualquer arquitetura. Para a implementação do sistema proposto foram ponderadas várias alternativas. Uma das alternativas seria o desenvolvimento de um sistema de raiz, mas, já existia um sistema desenvolvido que é o Notification Enabler. Este sistema implementado no projeto vf-OS, contém já algumas das características e requisitos determinados para o projeto.

Por fim, para complementar a apresentação de informação em tempo real é necessário um estudo da mesma. Pretende-se então a implementação de um componente de análise de dados. Esse componente será responsável pela exibição dos dados através de gráficos que permitam retirar conclusões acerca do estado da prateleira no presente, mas, também uma previsão do estado da prateleira no futuro.

3 CONCEITO GERAL DE PROJETO

O presente capítulo destina-se à apresentação do conceito geral da solução proposta. Tendo em conta que este projeto é realizado em colaboração com a empresa *KnowledgeBiz*[4]. Primeiramente será introduzido o cenário do caso industrial, onde vai ser possível entender melhor o grande objetivo. Devido á complexidade do projeto proposto, o mesmo foi dividido em três partes. Essas partes são a **física**, a ***cloud*** e a **aplicação**.

3.1 Caso de Estudo

Como forma de melhorar a expriência dos consumidores nas lojas físicas têm sido adoptadas novas tecnologias pelos donos das grandes marcas. Processos como o reabastecimento de inventário podem ser optimizados com o uso destas tecnologias, informando os armazéns através de dados em tempo real tanto da procura como da quantidade de inventários existente. Estas tecnologias podem beneficiar tanto o consumidor como os proprietários, permitindo-os gerir melhor os produtos com maior saída dos com menor saída. O cenário escolhido é uma prateleira inteligente constituída por sensores, onde o utilizador terá acesso a inúmeras informações como a quantidade de objetos na prateleira. Se a quantidade de objetos estiver a um nível crítico o utilizador pode ser informado através da implementação de regras, como pode ser observado na Figura 3.1.

[4] https://www.knowledgebiz.pt/

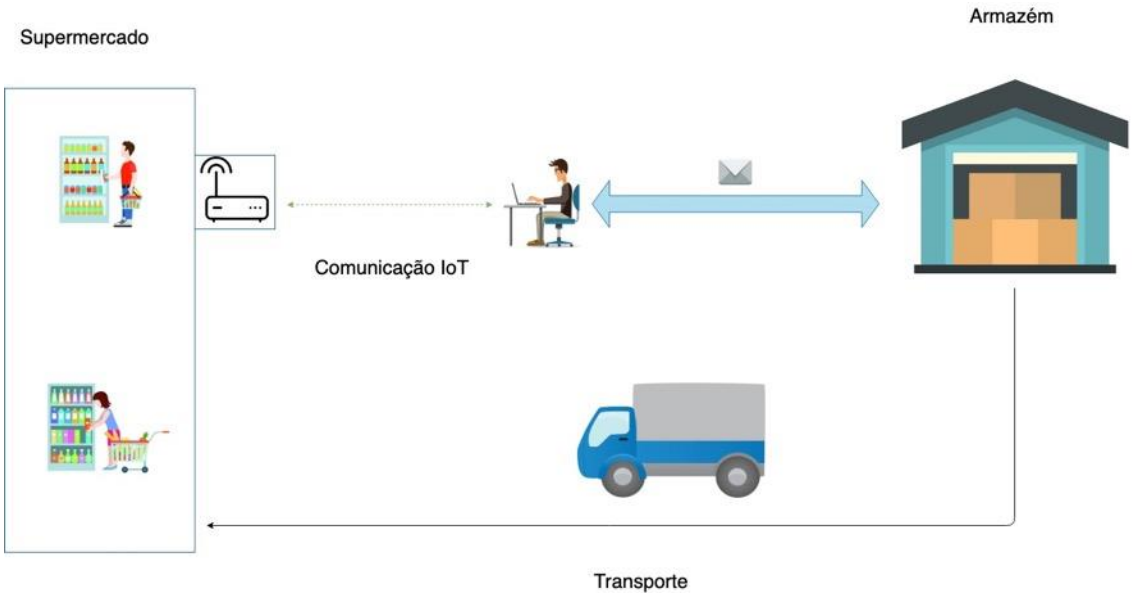


Figura 3.1 Diagrama do caso de estudo

3.2 Arquitetura Geral

O projeto engloba três partes que são a parte física, a *cloud* e a aplicação, como pode ser observados na Figura 3.2. Cada parte tem as suas funcionalidades bem definidas que serão descritas abaixo:

Parte Física: Esta parte é constituída por todos os dispositivos físicos(microcontroladores e sensores), desenvolvidos e implementados. Esses dispositivos são responsáveis pela recolha de dados, relativamente aos objetos colocado ou recolocados na prateleira.

Parte *Cloud*: Esta parte é constituídas pelos componentes responsáveis pelo armazenamento e processamento dos dados. Estes dados tanto podem ter origem nos dispositivos na secção Física como na secção da Aplicação.

Parte Aplicação: Esta parte é responsável pela interação com o utilizador, sendo, como a secção física, uma fonte de dados pois, existem funções de configuração além da apresentação de dados relevantes ao utilizador. Todos os dados recebidos nesta secção são enviados através do Middleware, que faz a ponte entre a parte Física e a *Cloud*.

Apesar da presente dissertação estar inserida neste projecto o grande foco da mesma é, a implementação de alguns componentes na secção *Cloud*. Estes componentes são o **Notification Manager** e o **Módulo de análise de dados** que estam assinalados na Figura 3.2 a tracejado.

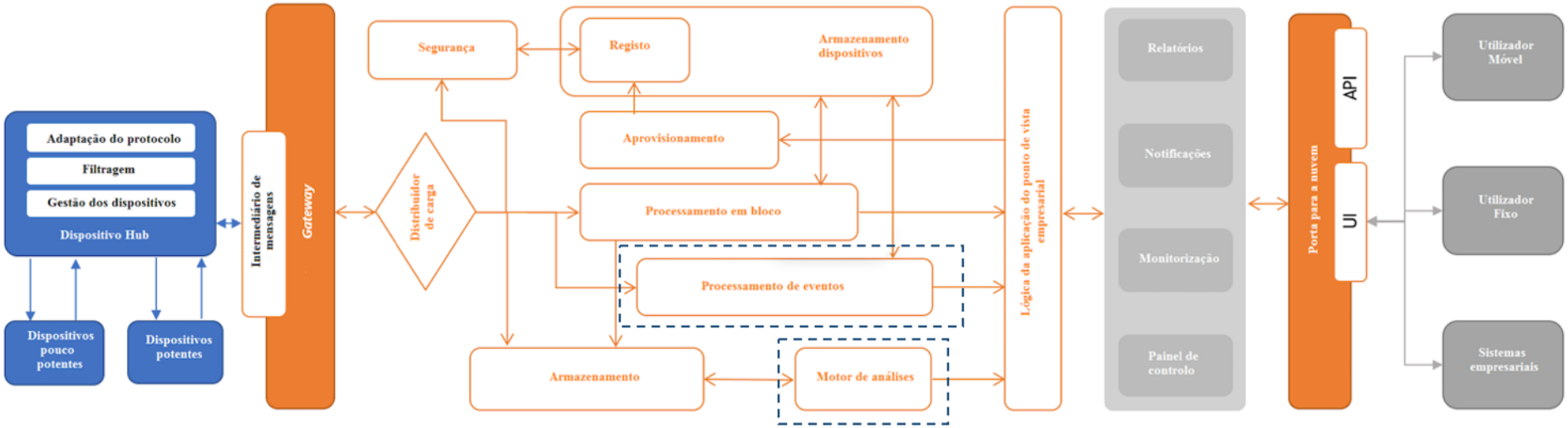


Figura 3.2 Arquitetura geral do projeto

3.3 Componentes a implementar

Como já havia sido anteriormente referido, a presente dissertação é responsável pela implementação do componente motor de validação de regras, denominado *Notification Manager*, e do Módulo de análise de dados. Com o primeiro é possível implementar e gerir regras que serão posteriormente aplicadas aos dados e assim produzir uma ação e/ou um resultado. O segundo componente é designado a análise descritiva estatística dos e visualização dos dados. Os dados provenientes dos dispositivos físicos, microcontroladores e sensores, são enviados pelo *middleware*, que é identificado por *Hub*, distribuindo os dados para os componentes inseridos na parte *cloud*. Os componentes desenvolvidos estão também interligados com um componente de armazenamento de dados que é o Repositório, fundamental para o bom funcionamento desta parte. A arquitetura modular dos componentes anteriormente referidos pode ser observada na seguinte Figura 3.3.

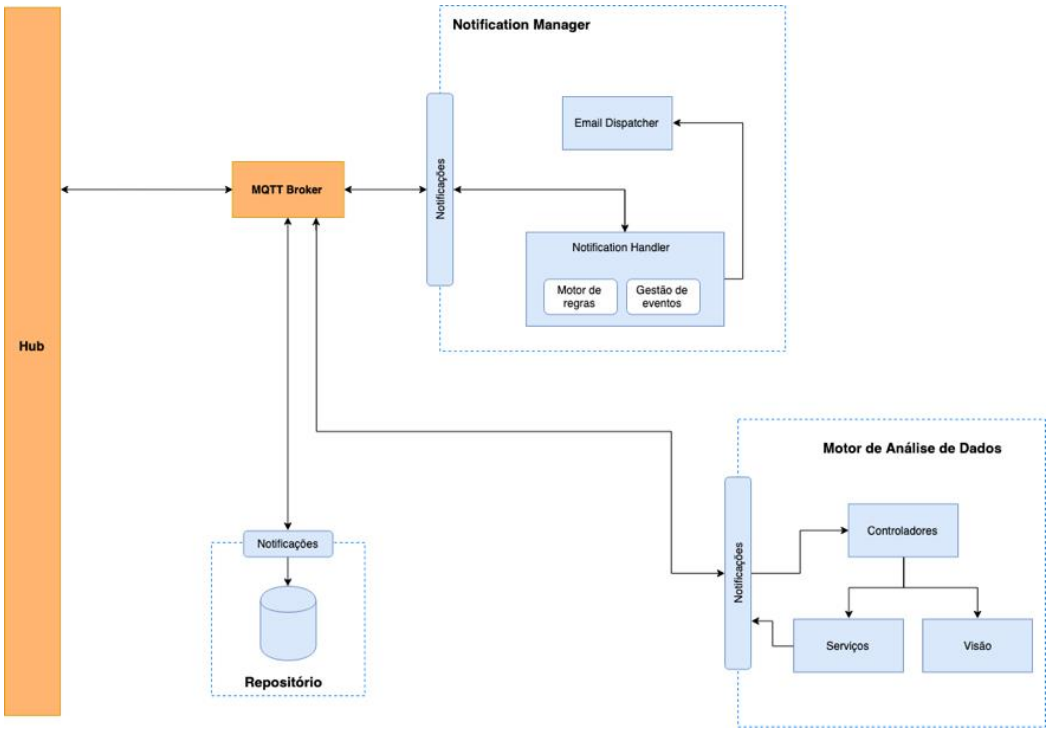


Figura 3.3 Arquitetura modular do sistema

O *Notification Manager* é constituído por três grandes módulos. O módulo principal é o Notifiation Handler onde se encontra o motor de regras, responsável pela comparação entre as regras e os dados, e a gestão de eventos que controla o envio de notificações consoante as conclusões do motor de regras. O *Email Dispatcher*, assim como o nome indica, tem como objetivo a preparação e envio das notificações via email. O último módulo são as notificações que são recebidas e depois encaminhadas para o *Notification Handler* onde são processadas.

O módulo de análise de dados é constituído por controladores, serviços e a visão. Os controladores são aonde os dados são tratados antes de serem apresentados ao utilizador através do módulo visão. Os serviços são simplesmente as funções que auxiliam na realização de pedidos tornando o código reutilizável. Estes módulos serão descritos com maior profundidade na secção 4.5.

O repositório é constituído por funções relacionadas com o tratamento e aquisição de dados armazenados por outros componentes. Os dados com origem na parte física chegam à *cloud* através do *middleware* e a comunicação entre todos os componentes é realizadas através de um *broker* MQTT que reencaminha as mensagens para os compontes subscritos aos tópicos.

4 IMPLEMENTAÇÃO

Neste capítulo serão descritas as ferramentas e bibliotecas utilizadas assim como as suas implicações no sistema. Também será discutida a metodologia de implementação do sistema, que foi dividido em três camadas: a camada de acessos aos dados, a camada lógica e a camada interface do utilizador. Cada um dessas mesmas camadas serão descritas no presente capítulo.

4.1 Tecnologia Utilizada

Os componentes que englobam o sistema foram desenvolvidos através da linguagem de programação Javascript[5] e estruturados com base no modelo cliente-servidor. Este modelo permite a distribuição de tarefas, sendo o servidor encarregue pela partilha de serviços e/ou funções para um ou mais clientes. Os clientes consomem esses recursos partilhados, sendo capaz de realizar processamento.

O servidor ou *backend*, como é também denominado, foi implementado com a tecnologia Node.js[6]. Esta tecnologia tem a particularidade de ser baseada em eventos, permitindo programação asíncrona, ou seja, o código continua a correr enquanto aguarda resposta de um pedido realizado. Esta capacidade possibilita o desenvolvimento de aplicações escaláveis.

O lado do cliente ou *frontend*, foi implementado com recurso ao Angular.js[7]. Apesar de ser baseada também na linguagem de programção Javascript, esta tecnologia utiliza também

[5] www.javascript.com
[6] https://nodejs.org
[7] https://angularjs.org

HTML[8] e CSS[9] no desenvolvimento de interfaces para o utilizador. A comunicação entre os componentes é conseguida através do protocolo MQTT com o envio de dados numa estrutura JSON e alguns desses dados serão apresentados em tempo real ao utilizador attravés do procotolo de Websockets.Esta estrutura foi conseguida com a utilização de bibliotecas open-source.

Na Figura 4.1 é realizado um levantamento das tecnologias e ferramentas utilizadas e indicação em que parte da estrutura cliente/servidor são utilizadas.

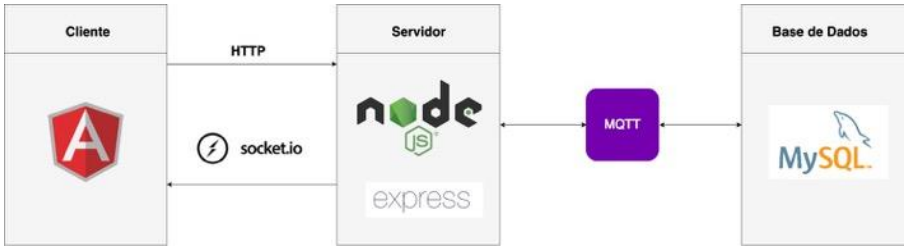


Figura 4.1 *Stack* de tecnologias e ferramentas

4.1.1 Bibliotecas *open-source*

- Eclipse Mosquitto – é uma implementação *open source* de um intermediário de mensagens do protocolo MQTT.

- Express.js – é uma simples biblioteca desenvolvida em cima do node.js com uma variedade de métodos HTTP para auxiliar na criação de API's.

- jQuery – é uma biblioteca Javascript versátil utilizada para simplificar a manipulação e gestão de eventos num ficheiro HTML, compatível com inúmeros navegadores *web*.

- MQTT.js – é uma biblioteca cliente desenvolvida em Javascript, baseada no protocolo de comunicação MQTT, para o node.js.

[8] https://www.w3.org/TR/html52/
[9] https://devdocs.io/css/

- Plotly.js – é uma biblioteca *open-source* de alto nível utilizada no desenvolvimento de ambientes gráficos dinâmicos e personalizáveis.

- Socket.IO – é uma biblioteca baseada numa arquitetura de eventos, que que permite a comunicação bidirecional e em tempo real entre o navegador *web* e o servidor.

4.2 Comunicação de dados

Para garantir a interação entre o *backend* e o *frontend*, troca de dados, foi implementada uma API. A API permite que a integração destes dois sistemas seja feita de uma forma simples. Esta API é baseada no estilo de arquitetura REST, baseado no protocol HTTP, que usa os métodos GET, POST, PUT e DELETE e possibilita a permuta de dados em formato JSON. O encaminhamento de pedidos e respostas por HTTP é implementado através da biblioteca Express.js.

A comunicação entre os componentes na *cloud*, como pode ser é vista na Figura 4.1, é realizada com o uso da tecnologia MQTT. Esta tecnologia é baseada na metodologia *publish/subscribe* e é importante pois viabiliza que o *Notification manager* consiga comunicar tanto com o *Middleware*, por onde recebe os dados dos dispositivos físicos, como com o componente responsável pelo armazenamento de dados. A implementação desta tecnologia do lado do cliente foi realizada através da biblioteca MQTT.js. O MQTT *broker* é uma das partes mais importantes porque é onde ocorre o processo de filtração das mensagens enviadas e para implementação do mesmo recorreu-se ao Eclipse Mosquitto.

Finalmente um dos requisitos deste componente é a alta interatividade e feedback visual imediato. Para garantir que esses requisitos são respeitados são necessárias tecnologias que possibilitam comunicação em tempo real entre o servidor e o cliente. A principal tecnologia vái ser por *Websockets*.

4.3 Funcionalidades implementadas

O sistema foi divido em várias camadas para facilitar a percepção do seu funcionamento e a divisão de tarefas do sistema. Essas camadas são:

- Camada interface de utilizador;
- Camada lógica;
- Camada de acesso a dados.

Na Figura 4.2 são descritas as funcionalidades gerais de cada uma das camadas referidas.

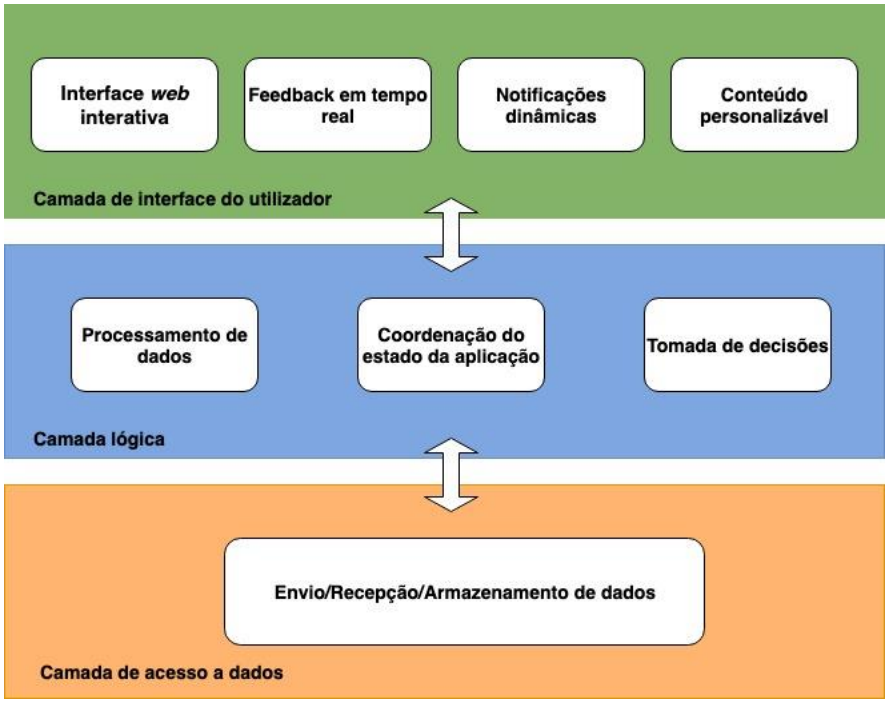


Figura 4.2 Estrutura funcional do sistema

4.4 Camada de acesso a dados

A camada de Acesso a Dados é constituída pelo componente responsável pelo armazenamento de dados que é o Repositório que está na secção *Cloud*. Este componente tem uma grande importância porque permite guardar os dados não só dos componentes na *Cloud* como os dados vindos do *Middleware*. Os dados têm como origem duas principais fontes o Middleware e o utilizador.

O desenvolvimento da base de dados situada nesse componente foi implementado com o sistema MySQL, que utiliza a tecnologia SQL. O sistema MySQL tem grandes vantagens como:

- Não é muito exigente ao nível do *hardware*
- Apresenta bom desempenho e estabilidade
- Fácil integração, pois, é compatível com várias linguagens de programação como C/C++, Python, Java e mais.

Como o Repositório tem o papel de armazenar os dados, sempre que necessários por outros componentes eles têm que ser pedidos. Essa interação é realizada através da tecnologia MQTT e a biblioteca MQTT.js, como já havia sido referido no capítulo 4.2.

Os pedidos realizados ao Repositório em formato JSON têm um esquema próprio, ilustrado na Figura 4.3. Além dos dados, devem conter os campos *Command* e *Replyto:*

- *command*: indica o serviço pedido a base de dados;
- *replyto*: indica o tópico MQTT para o qual deve ser enviada a resposta.

{
"**command**": *string*,
…
Dados
…
"**replyto**": *string*
}
	{
	"**command**": "retrievenotificationlist",
	"**hubid**": "node1",
	"**replyto**": "getnotificationlist"
	}

Figura 4.3 Esquema de exemplo de dados em formato JSON de pedidos ao repositório

Todos os dados enviados para o Repositório devem conter os campos *command* e *replyto*, respeitando o esquema definido na Figura 4.3, cada um dos campos tem a sua funcionalidade.

Sendo que a base de dados tem inúmeros serviços ou funções, sempre que recebe dados tem que ser informada que serviço foi pedido. Esses serviços podem ser, por exemplo, o armazenamento de dados relativos a um certo utilizador como pode ser um pedido de dados, exemplificado na Figura 4.4.

```
case "retrievenotificationlist":
        retrieveNotificationList(msgparsed.hubid, function (isOK, data) {
            if (isOK) {
            conn.publish(msgparsed.replyto, JSON.stringify(data));
            } else {
          conn.publish(msgparsed.replyto,JSON.stringify("error"+ data));
            }
        });
```

Figura 4.4 Exemplo de pedido de serviço à base de dados

Da mesma forma, a resposta pode conter dados ou uma simples confirmação de serviço bem executado. O processo de recepção de pedidos do Repositório pode ser observado no diagrama representado na Figura 4.5.

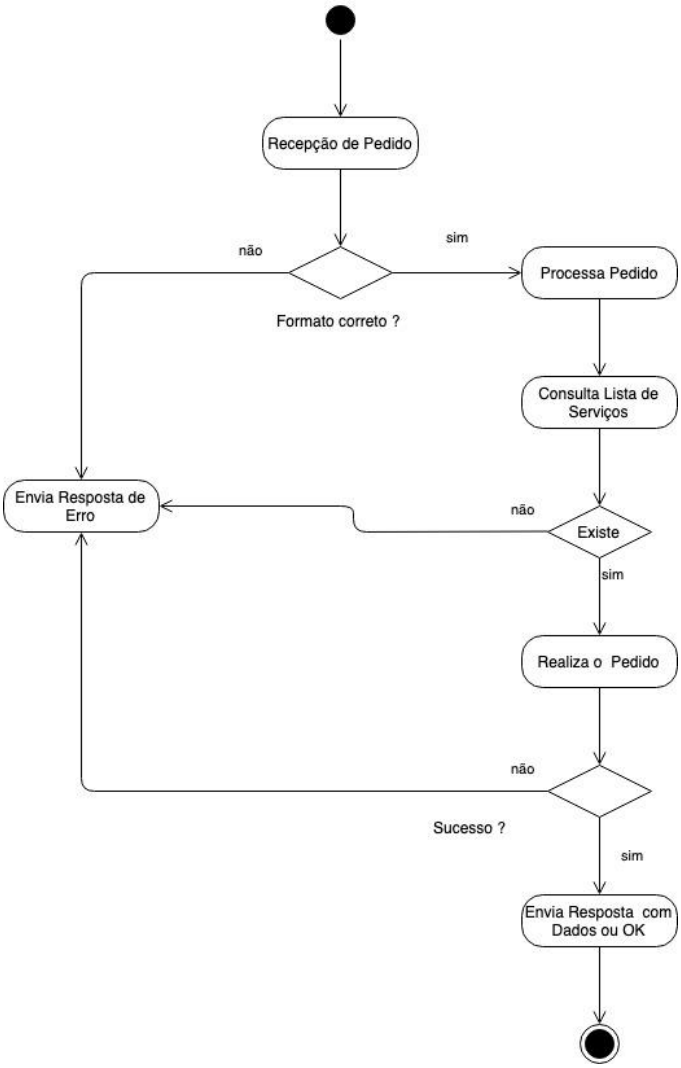


Figura 4.5 Diagrama de atividade do Repositório

4.4.1 Base de dados

O Repositório armazena os dados numa base de dados relacional em MySQL, como ilustrado na Figura 4.6. A base de dados é constituída pelas tabelas:

- Ruleslist, é a tabela responsável por armazenar as configurações das regras criadas pelo utilizador na UI;
- Statisticslist, é a tabela responsável pelo armazenamento de estatísticas em relação as regras;
- Notificationlist, é onde são armazenados os dados sujeitos a sujeitos a serem correspondidos pelas regras;
- Registration, é a tabela que armazena dados do utilizador;
- Messages, é onde são armazenadas as notificações de aviso para o utilizador;

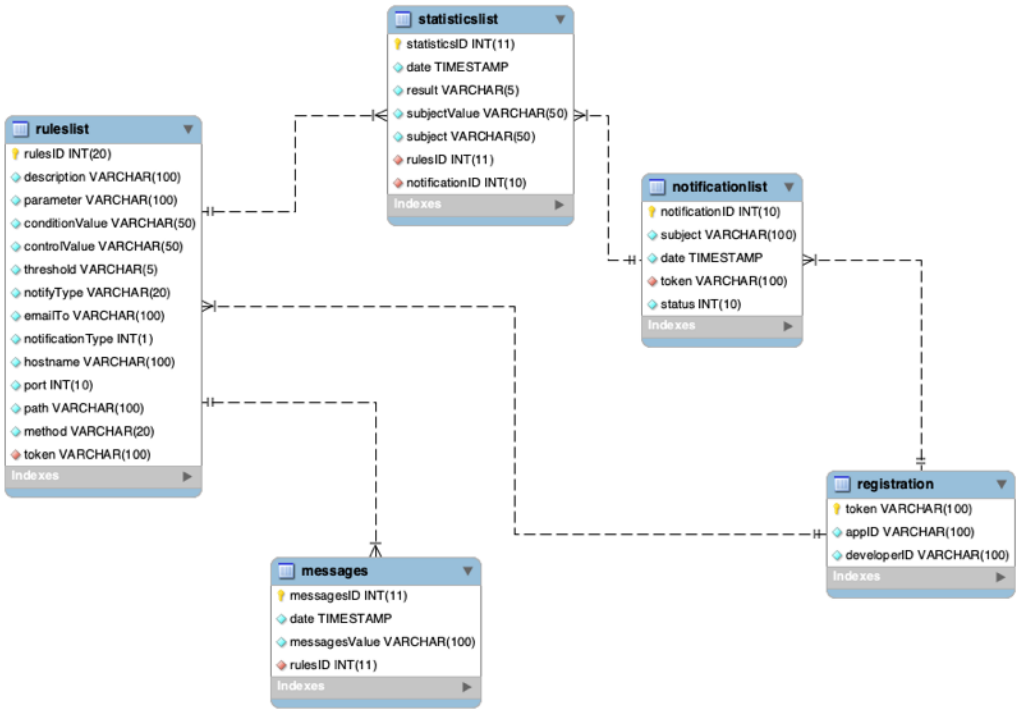


Figura 4.6 Diagrama ER

4.5 Camada de interface do utilizador

No presente subcapítulo vai ser descrito de uma forma mais pormenorizada a camada de interface do utilizador indicando a sua constituição, implementação e algumas secções do sistema, através de imagens.

A camada de interface do utilizador é constituída principalmente pelo módulo de análise de dados, tendo integrado em si o frontend do motor de validação de regras, os dois principais objetivos do módulo de análise de dados é a exposição e análise descritiva dos dados.

Uma interface de utilizador (UI) é a ponte encarregada pela interação entre o utilizador e computador através de páginas *web*. A página deve estar formatada com conteúdo relevante e facilmente perceptível. O formato e a organização das páginas vão ser descritas nesta secção. A UI do sistema da aplicação está dividida em três partes:

- Cabeçalho, é possível observar a identidade do utilizador e tem acesso rápido à página que das mensagens assim como indica também o número de mensagens recebidos.
- Barra lateral, onde estão indicadas todas as secções da aplicação web assim como a identidade do utilizador
- Secção principal, é onde pode ser observado todo o conteúdo relevante da aplicação web, mudando consoante a secção selecionada

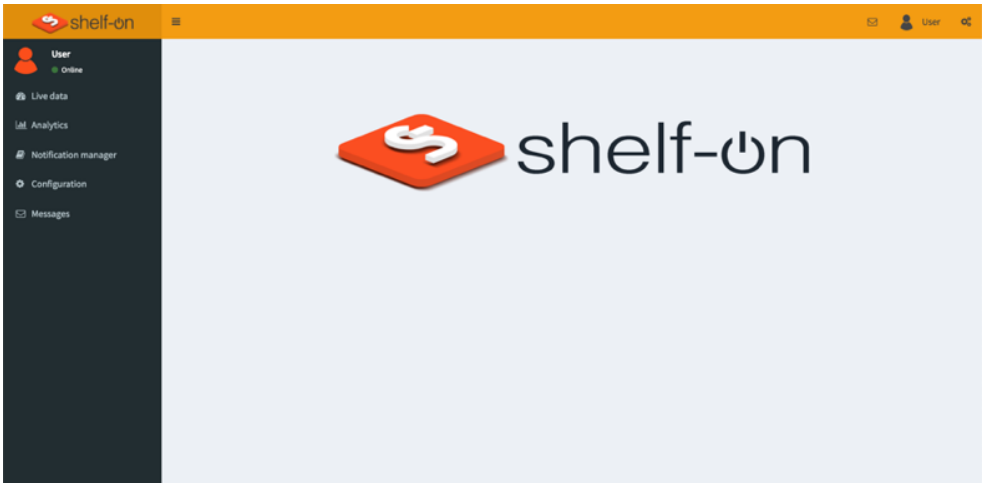


Figura 4.7 Página inicial

Na Figura 4.7 pode ser observada a página principal do sistema assim como as suas divisões, anteriormente descritas. No centro da secção principal estão ilustrados o logótipo e o nome *Shelf-On*[10].

O *frontend* de cada um dos componentes está integrado nesta camada e está dividido em componente 1 e componente 2. O componente um é o módulo de análise de dados e o componente 2 é o *notification manager*.

Cada uma das estruturas frontend é constituída por ficheiros javascript e HTML. Os ficheiros em javascript contêm os serviços e os controladores. Os serviços são uma forma de reutilizar e organizar o código em toda a aplicação. Os controladores têm simplesmente a lógica de cada página *web*. Os ficheiros HTML são relativos a estrutura e conteúdo das páginas web dos componentes. Cada página web tem associada a si um controlador, como por exemplo:

```
.when('/comp2/:developerid', {
        templateUrl: 'comp2/v/appList.html',
        controller: 'listCtrl'
    })
```

Figura 4.8 Exemplo do acesso a uma página HTML

No exemplo observado na Figura 4.8, sempre que for acessado o caminhdo /comp2/:developerid vai ser apresentada a página web representada pelo ficheiro appList.html.

[10] É o nome preliminar para o protótipo a ser desenvolvido na Knowledgebiz

Como pode ser observado no caminho, é composto pela palavra comp2 logo a página apresentada pertence ao *frontend* do componente dois.

Assim que está página for acessada será utilizado um serviço implementado através do Angular.js onde será efetuado um pedido HTTP. Normalmente este pedido seria através de um método *GET*, mas, como já havia sido referido antes, a base de dados está no componente responsável pelo armazenamento de dados. Este componente tem de ser acedido através do protocolo de comunicação MQTT. Então o pedido de dados da página será um método POST com a informação necessária para o Repositório. O processamento do Repositório a esta mensagem está descrito no capítulo 4.4.

```
getListByDevId: function (developerid) {
        var options = {
            topic: "db",
            command: "retrieveApp",
            developerID: developerid,
            replyto: 'listbydevid'
        };

        return $http.post('/api/proxy', {
            opt: options
        });
    },
```

Figura 4.9 Exemplo de serviço no *frontend*

Como pode ser observado na Figura 4.9 foi chamado o serviço getListByDevId. Esse serviço vai realizar um pedido HTTP com o método POST com um JSON com um tópico, um comando o ID do utilizador e o tópico para o qual deve ser enviada a resposta. O tratamento deste pedido será processado pela camada lógica de negócio e depois pela camada de acesso aos dados. A resposta a este pedido será através do protocolo de comunicação *Websockets* apesar de o pedido ter sido em HTTP.

Os dados emitidos por Webscoket são enviados diretamente para o controlador responsável pela página acessada e depois apresentados na página. Esses dados recebidos via pedido/resposta não serão os únicos enviados utilizando o protocolo Websocket, mas também os dados que vêm sem serem pedidos como por exemplo os dados enviados pelo Middleware. O envio dos dados pelo middleware é assíncrono e informação como o número de objetos tem de ser atualizada e apresentada em tempo real. O processamento deste comportamento assíncrono será descrito mais pormenorizadamente no próximo capítulo 4.6.

4.5.1 Secção Configuration

Nesta secção do sistema é onde ocorre a configuração do intervalo de tempo do envio de dados por parte do dispositivo físico. Também é possível a configuração do estado da prateleira como descrito na Tabela 4.1.

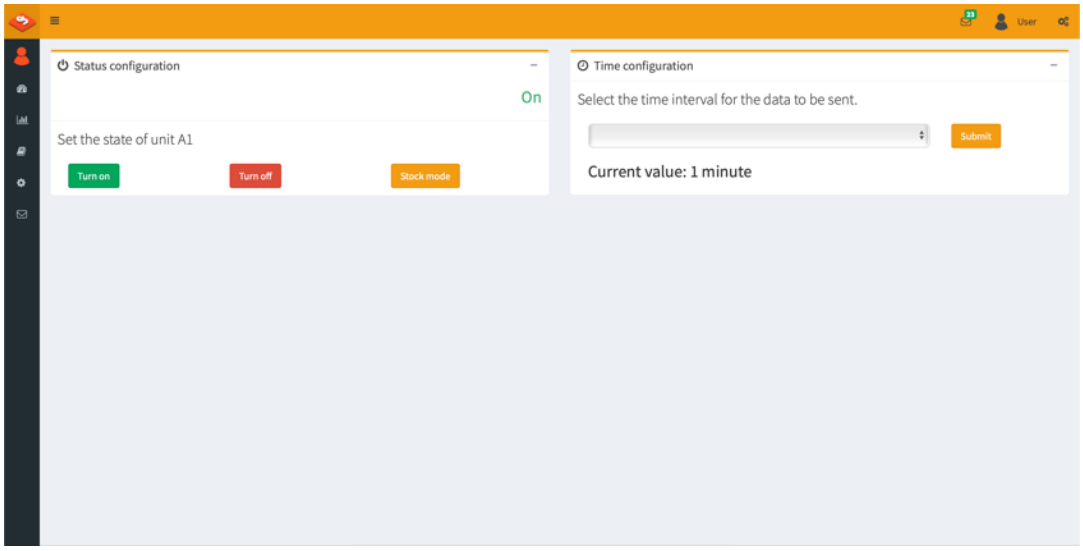


Figura 4.10 Secção configuration da aplicação web

4.5.2 Secção Live data

Nesta secção do sistema, é apresentada uma matriz com o posicionamento dos objetos, o número de objetos e a data da última atualização. Tendo em conta o caso ilustrado na Figura 3.1, a matriz é criada com informações enviadas pelo sensor.

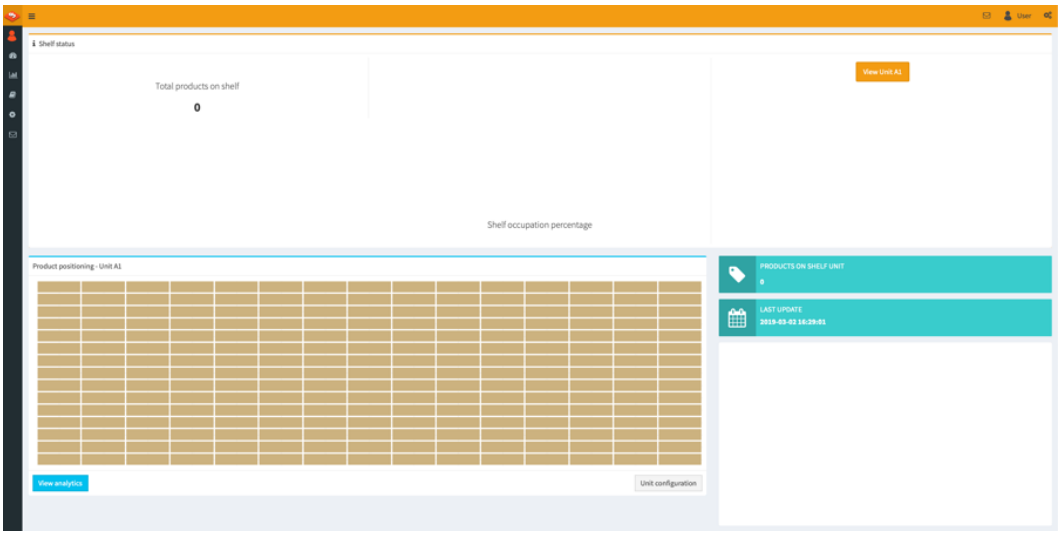


Figura 4.11 Secção live data da aplicação web

O sensor, com um formato de matriz, recolhe todos as posições onde existe pressão. Essas posições são registadas com o valor de 1 e onde não existe pressão o valor é 0 como pode ser observado na Figura 4.12. Após o processamento através de dispositivos físicos, essa informação é encaminhada pelo *middleware* para os componentes *cloud*.

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	1	1	1	0	0	0	0	0	0	0	0	0
0	0	0	1	1	1	0	0	0	0	0	0	0	0	0
0	0	0	1	1	1	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	1	1	0	0	0	0	0
0	0	0	0	0	0	0	0	1	1	0	0	0	0	0
0	0	0	0	0	0	0	0	1	1	0	0	0	0	0
0	0	0	0	0	0	0	0	1	1	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figura 4.12 Matriz com o posicionamento dos objetos

4.5.3 Notification Manager

Nesta secção do sistema podem-se encontrar as páginas web do componente Notification Manager. Na primeira página é ilustrado o id do sensor, id do *developer* e alguns botões para

aceder às notificações recebidas e as regras configuradas. O developer é a pessoa responsável pela gestão do sistema. Existe também um botão para a configuração de um novo sensor e um para apagar todos os dados do sensor configurado.

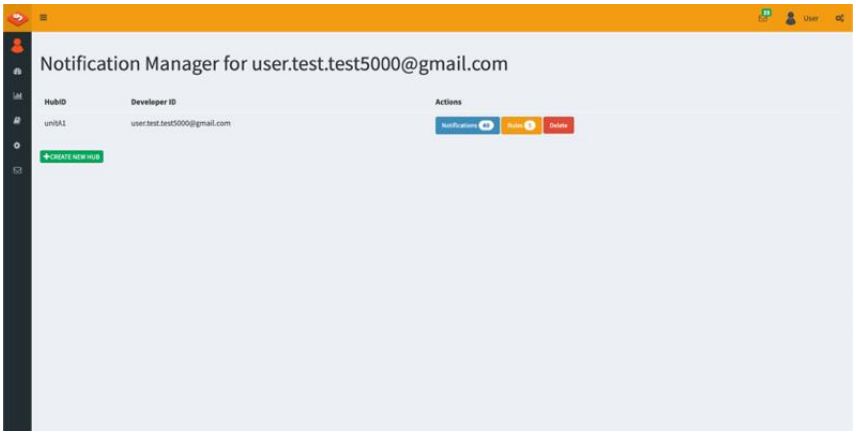


Figura 4.13 Página inicial do Notification manager

Selecionado o botão Rules o utilizador é direcionado à página com as regras configuradas para o sensor. As regras são constituídas por vários campos:

- *RulesID*, é o número de identificação da regra;
- *Description*, é uma breve descrição da regra;
- *Parameter*, é a identificação do tipo da regra;
- *Condition*, é o operador relacional para a comparação de valores;
- *Control Value*, é o valor, inteiro, que será comparado;
- *Threshold*, é o valor, em percentagem, que será comparado;
- *NotifyType*, tipo de notificação de alerta.

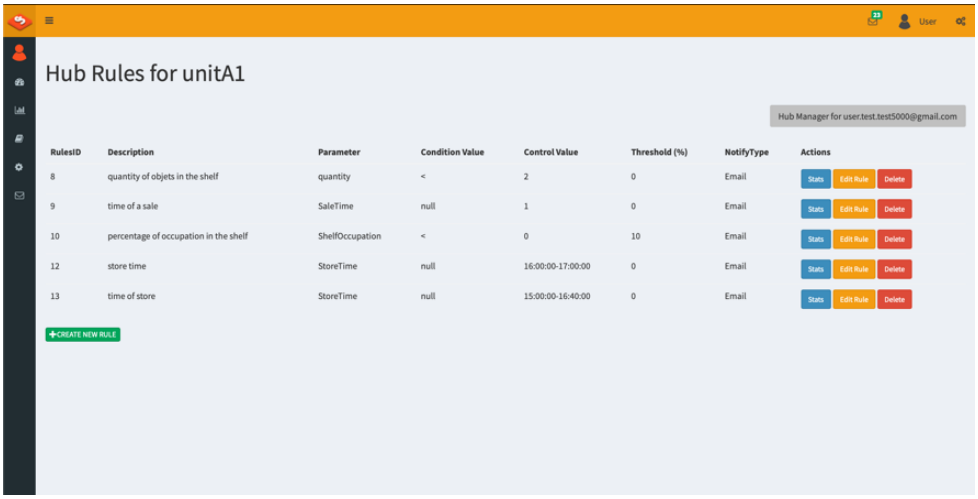


Figura 4.14 Regras configuradas

Na página ilustrada na Figura 4.14, podem-se verificar ainda alguns botões para executar ações. Essas ações podem ser a visualização de estatísticas, alteração da regra, eiminação da regra ou a criação de uma nova regra. Após selecionar o botão para a criação de uma nova regra é apresentada a página observada na Figura 4.15. Nessa figura é possível observar a configuração bem-sucedida de uma regra.

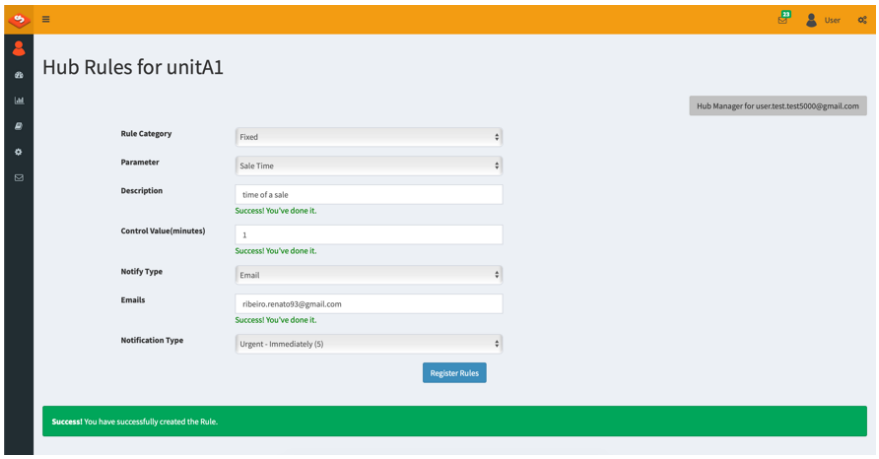


Figura 4.15 Exemplo de criação de uma regra da categoria fixed para a taxa de ocupação da prateleira

4.5.4 Secção Messages

Nesta secção da aplicação web é onde podem ser observadas os alertas recebidos pelo utilizador em relação às regras configuradas.

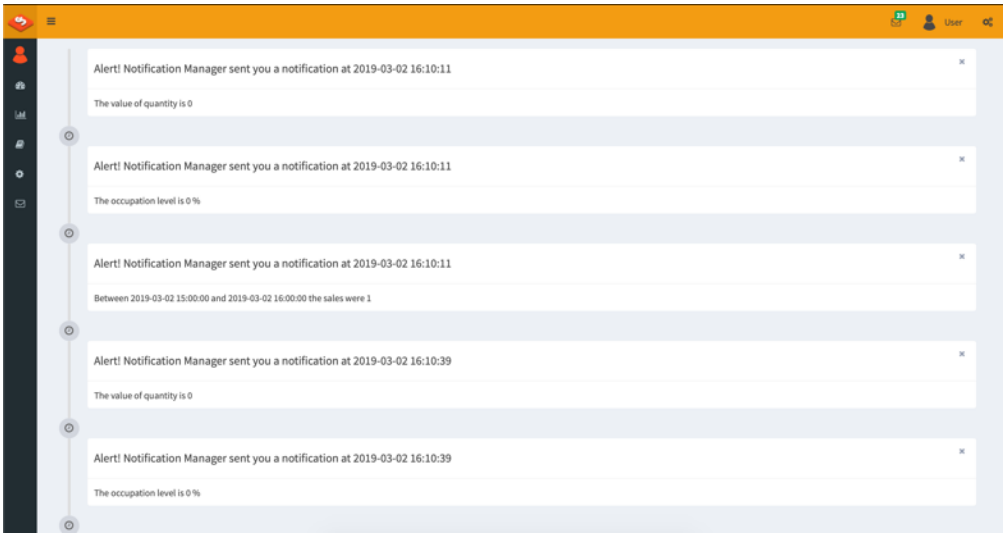


Figura 4.16 Secção messages da aplicação web

4.5.5 Secção Analytics

Nesta secção do website pode-se observar um gráfico com a evolução das vendas ao longo do tempo. Sendo que estamos perante um caso industrial, representado na Figura 3.1,de forma a assegurar que a reposição de objetos é executada de forma mais eficiente possível e correta é fundamental controlar a saída dos objetos. Sempre que o objeto esteje ausente mais do que um certo tempo configurado pelo utilizador será considerado como uma venda, existirá uma atualização no gráfico ilustrado na Figura 4.17.

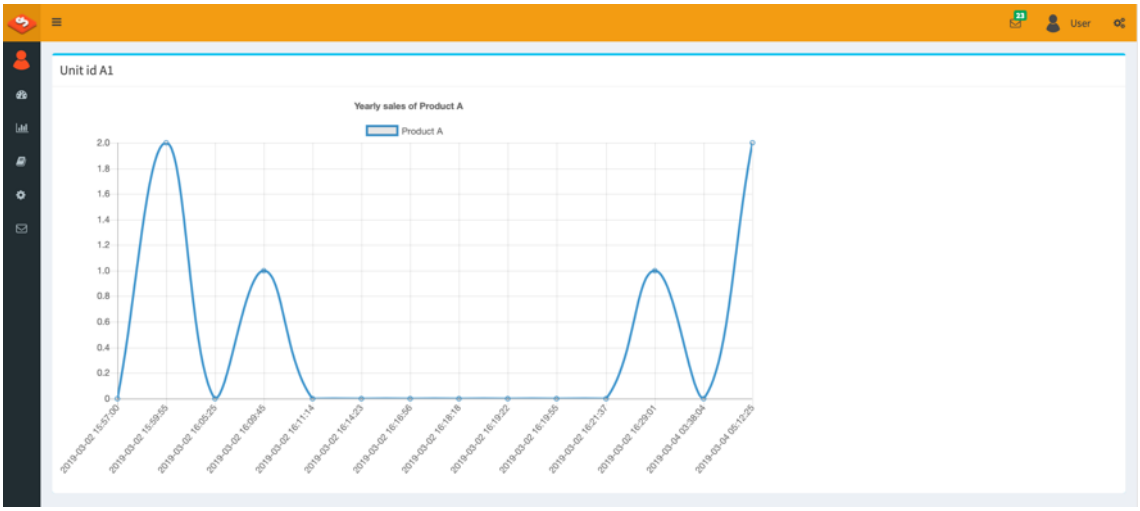


Figura 4.17 Secção analytics da aplicação web

4.6 Camada lógica

A camada lógica é o núcleo da aplicação, isto é, todos os dados passam por esta camada para serem processados antes de serem enviados tanto para a camada UI como para a camada de acesso a dados. Esta camada é constituída pelo *backend* dos componentes apresentados na presente dissertação. Sendo que o *backend* do Motor de análise de dados não faz grande processamento nem tem grande inteligência este capítulo focar-se-a mais no *Notification Manager*.

O grande foco desta camada é tomar decisões com base nos dados recebidos. Na é ilustrado o esquema dos dados enviados pelo middleware. Esses dados contêm:

- hubid: é o id do emissor/fonte dos dados;
- date: a data de quando os dados foram enviados;
- matrix: a representação da matrix com valores de 0 ou 1 em cada posição;
- objNumber: o número de objetos na prateleira;
- status: este é o estado da prateleira.

{
"body":{
"hubid": string,
"date": string,
"matrix :string,
"sizeX": int,
"sizeY": int,
"objNumber": int,
"status": int
}
}
	{
	"body": {
	"hubid" : "node1",
	"date":"2019-02-013T13:53:10.000Z",
	"matrix":"0,0,0,",
	"sizeX": 14,
	"sizeY": 14,
	"objNumber": 40,
	"status": 1,
	}
	}

Figura 4.18 Versão simplificada do JSON enviado pelo Middleware

O estado da prateleira que varia entre três valores 0,1 e 2.

Tabela 4.1 Modos de funcionamento da prateleira

Status	**Modo**
0	offline
1	online
2	reposição

O modo offline permite identificar quando o sensor não se encontra operacional por algum problema ou foi desconectado. O modo online ocorre quando sensor está operacional e o modo reposição occore quando a prateleira está em reposição, ou seja, estão a ser colocados objetos que não estão a ser retornados não contando para as vendas.

Após a o pré-processamento dos dados a validade do id do emissor tem de ser verificada. Isto é, o motor de regras só processa dados enviados emissores que já tenham sido configurados na UI. Posteriormente são pedidas todas as regras respeitantes ao emissor de dados e depois são comparadas com a informação recebida. Se existir correspondência é tomada uma ação que neste sistema é uma notificação. Essa notificação pode ser um e-mail ou um aviso na interface de utilizador. Todos estes passsos podem ser observados no fluxograma da camada lógica de negócios da Figura 4.19.

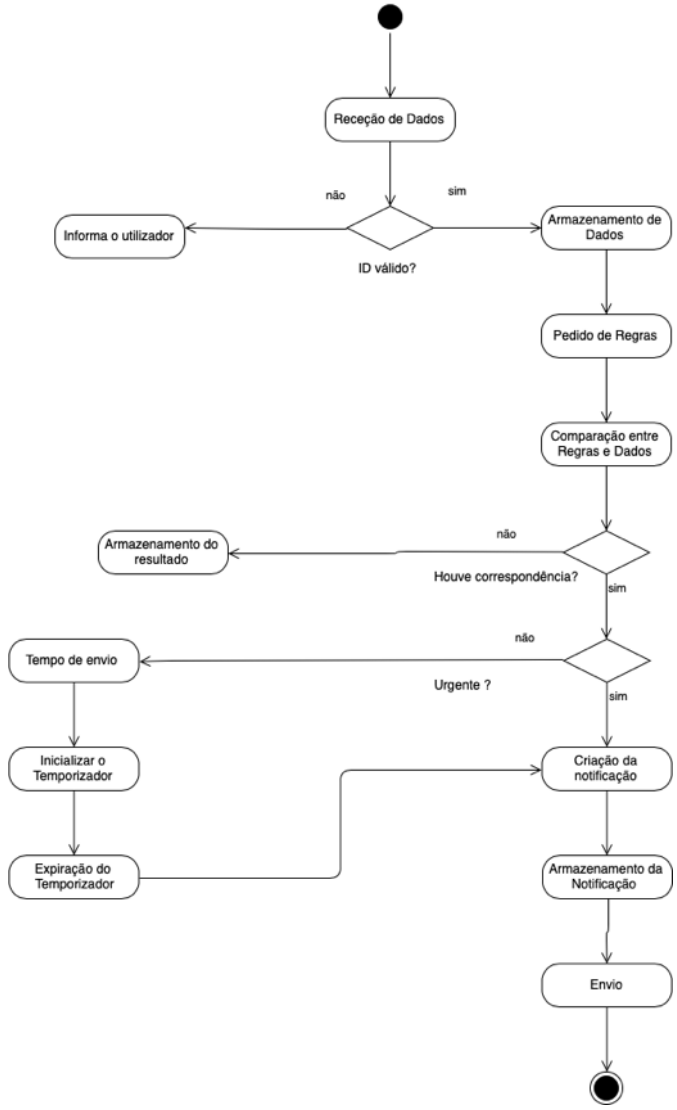


Figura 4.19 Fluxograma da camada lógica

4.6.1 Implementação de regras

Sendo que uma das grandes vantagens de um sistema de validação de regras (BRMS) é validação de regras ser expressa de uma maneira legível ao ser humano, a implementação será assente nesse princípio.

As regras podem ser constituídas por vários campos como podem ser observados na tabela *ruleslist* presente na Figura 4.6. A grande distinção entre as regras é o campo *parameter*. Este campo identifica a regra e pode ser personalizado ou não consoante a categoria da regra. Neste sistema as regras foram divididas em duas categorias: as regras *fixed*(fixas) e as regras *custom*(personalizáveis).

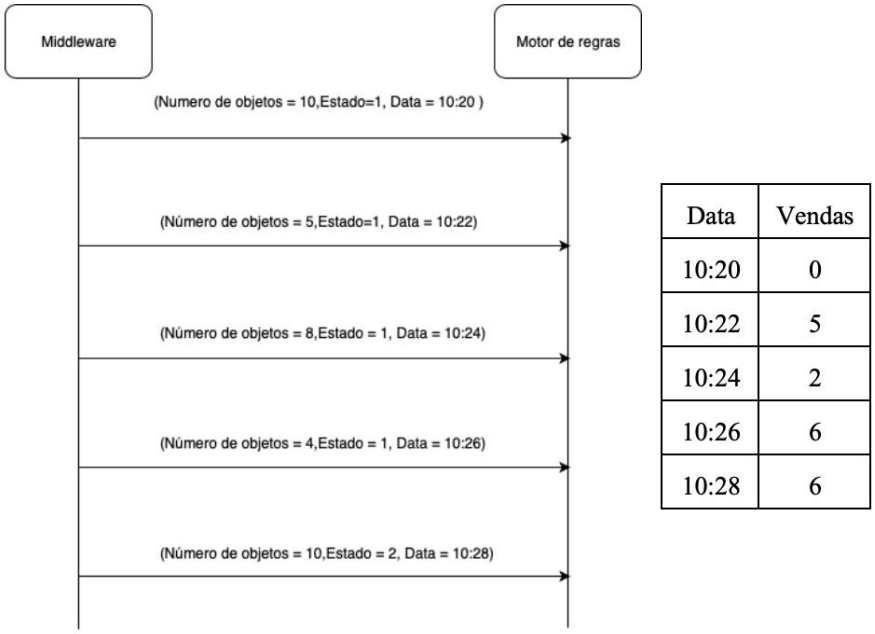
As regras *custom* podem ser configuras com um *parameter* á escolha do utilizador, daí serem regras personalizáveis. Estas regras funcionam com a comparação entre os valores numéricos presentes nos dados e o valor definido como Control Value.

Essa comparação é feita através do ConditionValue. O Condition Value é simplesmente um operador relacional responsável por comparar valores.

Sendo que este sistema está enquadrado num cenário industrial, a monitorização de uma prateleira situada numa superfície comercial, surgiu a necessidade da criação de regras mais específicas adaptadas a esse ambiente. Essas regras são as regras *fixed*. As regras fixed têm *parameters* fixos que são:

- *Sale Time*, onde é indicado tempo para ser considerada uma venda (em minutos);
- *Shelf occupation,* percentagem de ocupação da prateleira;
- *Store Time*, retorna o número de objetos vendidos durante um certo intervalo de tempo.

Para a melhor compreensão do conceito destas regras foi criado o diagrama presente na Figura 4.20. Neste exemplo é simulado o envio de dados para o motor de regras. Este exemplo é baseado numa regra de Sale Time com Control Value de 2.



Data	Vendas
10:20	0
10:22	5
10:24	2
10:26	6
10:28	6

Figura 4.20 Diagrama do funcionamento das regras fixed

Na tabela ao lado do diagrama podem ser vizualizados os números de objetos que foram vendidos consoante os dados recebidos. Nas duas últimas datas pode ser observado que o valor de vendas mantevesse. Isto devido ao estado da prateleira que está com o valor de dois, indicando

que a prateleira está em modo de utilização dois, ou seja, houve reposição de objetos. Esta reposição não conta para a venda. Esse modo foi descrito na Tabela 4.1.

4.6.2 Interação interface de utilizador/dados

Como já foi referido antes, o backend do motor de validação de regras tem um papel fundamental na coordenação do envio de dados entre os componentes da camada de acesso aos dados e os componentes da camada de interface de utilizador. Todos os pedidos realizados pelos componentes da camada UI são realizados através do protocolo HTTP e do método POST que é apresentada na Figura 4.9. Esses pedidos são depois reencaminhados via MQTT para o componente de armazenamento de dados que envia a resposta via o mesmo protocolo para o backend do motor de validação de regras. Um exemplo de um pedido pode ser observado na Figura 4.3. O backend do motor de regras recebe os dados e encaminhaos via Websocket para os componentes presentes na camada de interface do utilizador. Um exemplo de como esse processo é realizado é apresentado na Figura 4.21:

```
case 'hubnotification':
        io.sockets.emit('hubnotification',{data:JSON.parse(message)});
        break;
```

Figura 4.21 Encaminhamento de dados recebidos via mqtt para2w via websocket

A figura anteriormente apresentada demonstra a recepçao de dados recebidos no tópico mqtt “hubnotification” e depois enviado para o tópico “hubnotification” via websocket. Os dados serão processados nos ficheiros javascript do frontend e depois apresentados na interface se for o caso.

5 TESTES E VALIDAÇÃO DE RESULTADOS

No presente capítulo será introduzida uma análise ao sistema implementado. Primeiramente será descrito um cenário com o uso dos disposistivos físicos implementados para o projeto. Deseguida foram realizados ensaios em várias situações diferente que servem como base a validação dos testes. O sistema implementado foi denominado **ShelfOn.**

5.1 Cenário prático

Neste subcapítulo pretende-se apresentar o teste do sistema com o auxilío do protótipo, tentando recriar ao máximo situações reais de interação entre os objetos e clientes numa superfície comercial. A presente dissertação não é responsável pelo desenvolvimento do protótipo físico. As imagens utilizadas nesta subcapítulo do protótipo físico servem simplesmente para provar o funcionemento do sistema implementado.

Num primeiro teste foram colocados três objetos muito idênticos no protótipo, como pode ser observado na Figura 5.1.

Figura 5.1 Objetos colocados no protótipo

Após a colocação dos objetos no protótipo existe um processamento na secção física e no *Middleware* que de seguida envia os dados, que podem ser observados na Figura 5.2, para a secção *Cloud*.

{
"body":{
"hubid":" unitA1":,
"date":"2019-03-02T16:21:32.000Z",
"matrix :"
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 ,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 ,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0 ,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0 ,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0 ,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 ,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 ,
0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0 ,
0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 0 ,
0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 0 ,
0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 0 ,
0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 1, 0 ,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0 ,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0 ,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 “,
"sizeX":14,
"sizeY":14,
"objNumber":3,
"status":1
}
}

Figura 5.2 Exemplo de dados enviados pelo Middleware em formato JSON com a identificação de três objetos.

Assim que os dados são recebidos pelo sistema são apresentados ao utilizador em tempo real. Na Figura 5.3 pode-se observar a matrix com o posicionamento dos objetos no protótipo.

Figura 5.3 Matriz com o posicionamento de três objetos

5.2 Validação de testes

Após o cenário descrito na secção 5.1 com a utilização do protótipo, vai ser apresentada a validação de testes. A validação está organizada de modo a apresentar dados de entrada, configuração de regras e os dados á saída do sistema. O grande objetivo é testar o sistema garantindo que é uma solução viável.

5.2.1 Teste com um objeto

Neste primeiro teste foram configuradas três regras, que podem ser observadas na Figura 5.5. Cada regra contém a sua descrição, no campo *Description*, para ser perceptível a sua finalidade. Neste teste é esperado que se o número de objetos seja menor que dois e o valor da taxa de ocupação seja menor que dez por cento o utilizador receberá uma mensagem, alertando o acontecimento destes eventos.

- **Resultado esperado**

Como resultado deste teste são expectáveis duas notificações de alerta referentes á taxa de ocupação da prateleira e a quantidade de objetos.

- **Dados de entrada**

Na Figura 5.4 Exemplo de dados enviados pelo Middleware em formato JSON com a identificação de um objeto. é apresentada a notificação enviada pelo *Middleware*, que contém os dados relativos ao posicionamento do objeto. Através do campo *objNumber* é possível observar o número de objetos colocados na prateleira.


```
{
"body":{
"hubid":" unitA1":,
"date":"2019-03-02T16:29:01.000Z",
"matrix":
 " 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 ,
  0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 ,
  0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 ,
  0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 ,
  0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 ,
  0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 ,
  0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0 ,
  0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0 ,
  0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 ,
  0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 ,
  0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 ,
  0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 ,
  0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 ,
  0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 ",
"sizeX":14,"sizeY":14, "orientation":"left/right","objNumber":1
  }
}
```


Figura 5.4 Exemplo de dados enviados pelo Middleware em formato JSON com a identificação de um objeto.

- **Configuração de regras:**

Description	Parameter	Condition	Control Value	Threshold (%)
Tempo(minutos) para objeto ser considerado como vendido	SaleTime	null	1	0
Número de objetos na prateleira	quantity	<	2	0
Percentagem de ocupação da prateleira	ShelfOccupation	<	0	10

Figura 5.5 Regras configuradas para o primeiro teste

- **Resultados obtidos:**

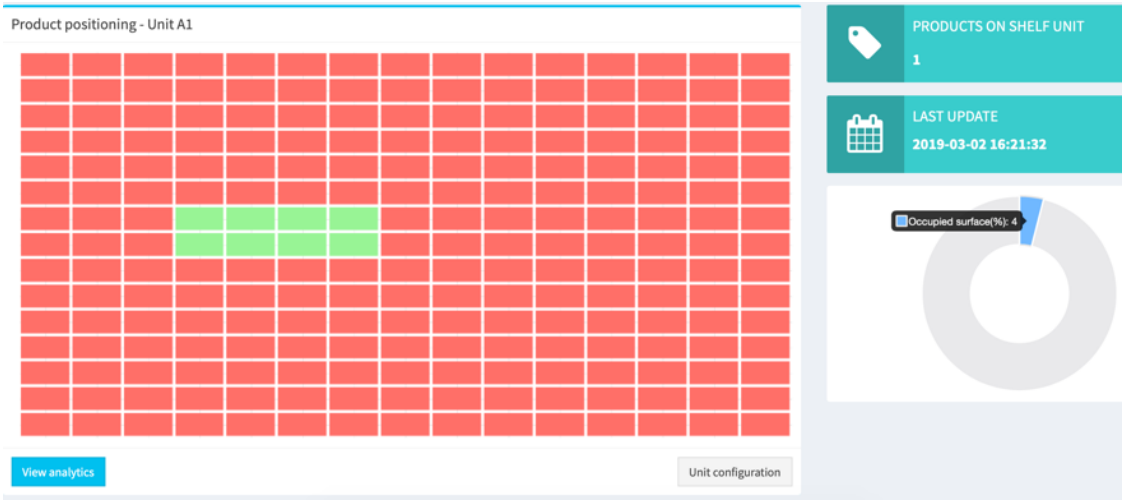


Figura 5.6 Matriz com o posicionamento de um objeto

Como seria esperado a taxa de ocupação foi muito menor em comparação á Figura 5.3 pois, neste caso só foi colocado um objeto . Como pode ser observado na imagem a taxa de ocupçao é de 4%. Tanto o número de objetos como a taxa de ocupação estão a abaixo dos valores impostos na regras, ou seja, existe correspondência e o utilizador é notificado. Essas notificações podem ser observada na figura seguinte:

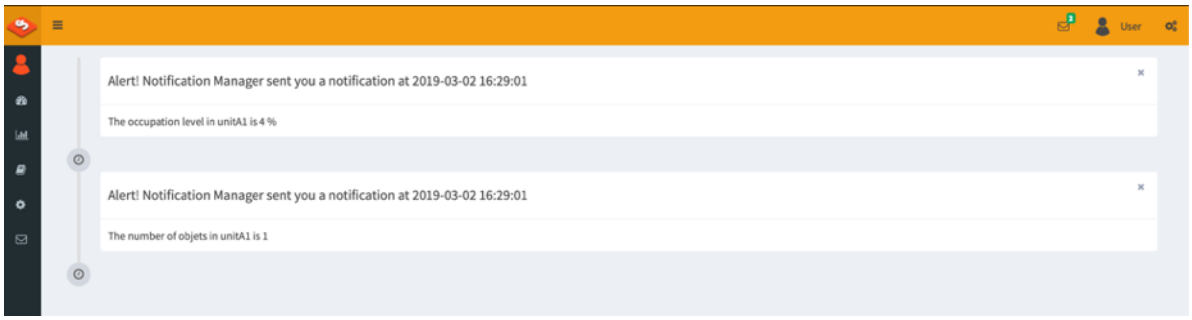


Figura 5.7 Notificações alerta na interface do utilizador

5.2.2 Teste com análise de vendas

Neste teste foram enviadas ao sistema, notificações com o número de objetos durante uma hora. Essas notificações foram enviadas com uma variação do estado, como pode-se observar na Figura 5.8. Como já havia sido indicado no subcapítulo 4.6, o funcionamento consoante o valor do estado é diferente.

- **Resultado Esperado**

Neste teste é esperado como resultado, uma representação gráfica das vendas entre o horário definido nas regras. No presente ensaio sempre que o valor é igual a dois o número de objetos não conta para o cálculo das vendas, ou seja, esse número de objeto não influência o gráfico de vendas. O número de vendas é depois apresentado como forma de notificação.

Dados de entrada:

Subject	Date	Status
quantity = 20	2019-03-15T00:23:29.000Z	1
quantity = 15	2019-03-15T00:24:34.000Z	1
quantity = 17	2019-03-15T00:25:22.000Z	1
quantity = 17	2019-03-15T00:26:10.000Z	1
quantity = 3	2019-03-15T00:29:37.000Z	1
quantity = 10	2019-03-15T00:30:17.000Z	2
quantity = 20	2019-03-15T00:30:40.000Z	2
quantity = 13	2019-03-15T00:31:19.000Z	1
quantity = 11	2019-03-15T00:31:51.000Z	1
quantity = 10	2019-03-15T00:32:08.000Z	1
quantity = 10	2019-03-15T00:32:35.000Z	1
quantity = 7	2019-03-15T00:32:50.000Z	1
quantity = 4	2019-03-15T00:33:18.000Z	1
quantity = 6	2019-03-15T00:33:53.000Z	1
quantity = 6	2019-03-15T00:34:57.000Z	1
quantity = 3	2019-03-15T00:35:43.000Z	1
quantity = 1	2019-03-15T00:36:52.000Z	1
quantity = 20	2019-03-15T00:37:51.000Z	2
quantity = 16	2019-03-15T00:40:53.000Z	1
quantity = 14	2019-03-15T00:41:34.000Z	1
quantity = 12	2019-03-15T00:44:13.000Z	1
quantity = 5	2019-03-15T00:45:23.000Z	1
quantity = 2	2019-03-15T00:46:54.000Z	1
quantity = 2	2019-03-15T00:56:10.000Z	1
quantity = 1	2019-03-15T00:57:38.000Z	1
quantity = 1	2019-03-15T01:00:09.000Z	1

Figura 5.8 Notificações do teste 2

- **Configuração de regras:**

RulesID	Description	Parameter	Condition	Control Value	Threshold (%)
37	Tempo(minutos) para objeto ser considerado vendido	SaleTime	null	1	0
38	Número de vendas entre um certo horário	StoreTime	null	00:00:00-01:00:00	0

Figura 5.9 Regras configuradas para o teste 2

- **Resultados:**

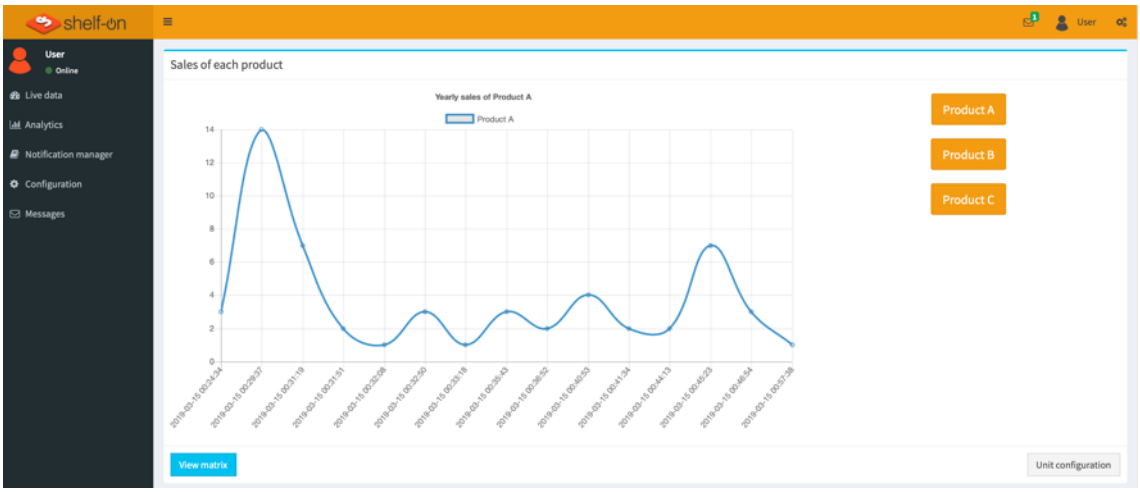


Figura 5.10 Evolução das vendas durante o teste 2

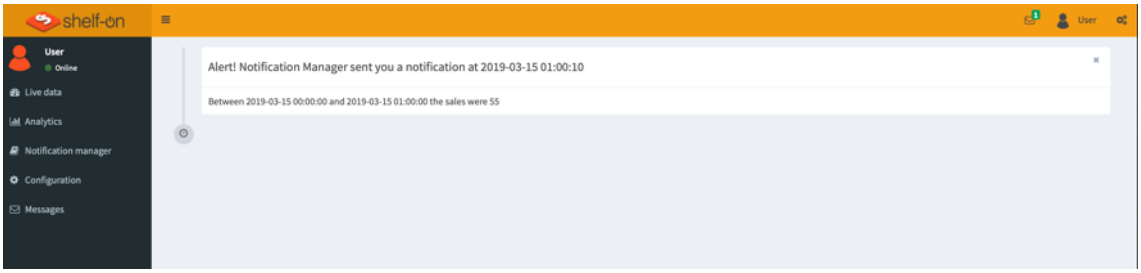


Figura 5.11 Notificação alerta com o número de vendas durante o teste 2

Como pode ser analisado através das imagens anteriores, o resultado foi o esperado. Tanto a evolução das vendas está corretamente representada como resultado do número de vendas era o esperado.

5.2.3 Teste com três prateleiras

Este segundo teste vai ser realizado com a configuração de três prateleiras no sistema. Os dados enviados a cada uma das prateleiras serão os mesmos, mas, com regras diferentes. O objetivo é provar que o sistema consegue trabalhar com este envio/recepção de dados simutâneamente.

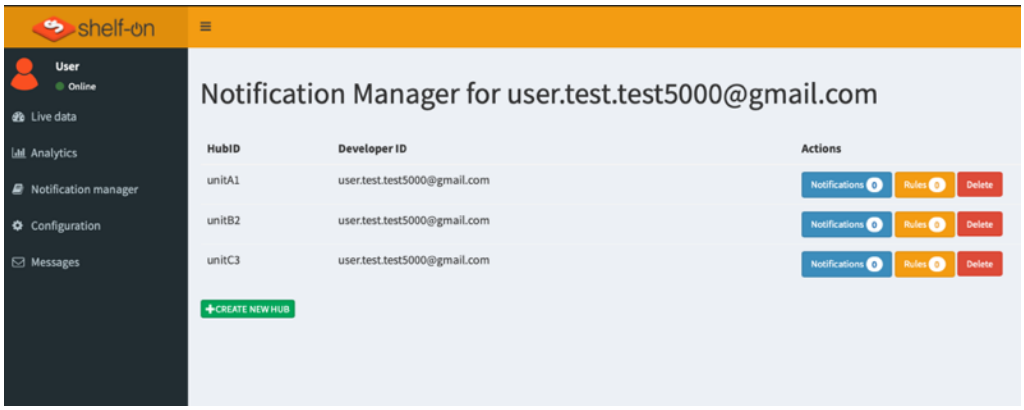


Figura 5.12 Prateleiras configuradas

- **Resultado esperado:**

Neste teste são expectáveis onze notificações de alerta. Dessas mesmas onze notificações são esperadas quatro notificações em relação a prateleira unitA1 são 4, da prateleira uniB2 são seis e da prateleira unitC1 é somente uma.

- **Dados de entrada:**

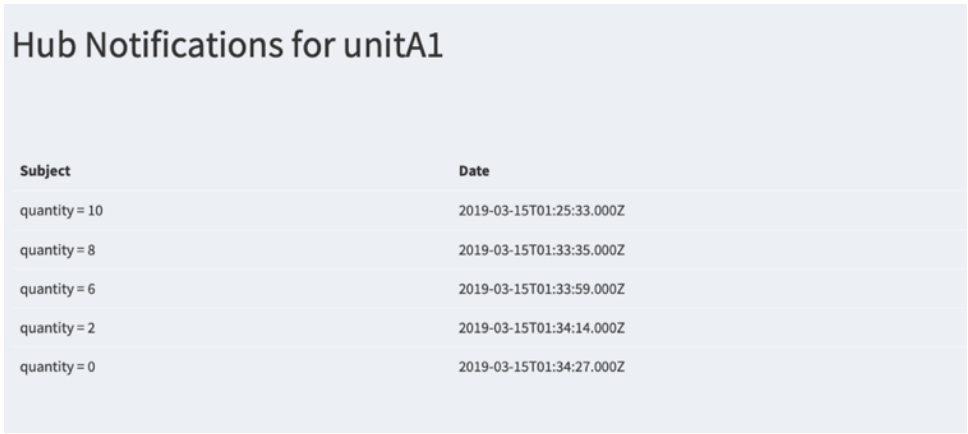


Figura 5.13 Notificações enviadas ao hub unitA1

Hub Notifications for unitB2

Subject	Date
quantity = 10	2019-03-15T01:25:45.000Z
quantity = 8	2019-03-15T01:38:20.000Z
quantity = 6	2019-03-15T01:38:33.000Z
quantity = 2	2019-03-15T01:38:46.000Z
quantity = 0	2019-03-15T01:38:58.000Z

Figura 5.14 Notificações enviadas ao hub unitB2

Hub Notifications for unitC3

Subject	Date
quantity = 10	2019-03-15T01:25:56.000Z
quantity = 8	2019-03-15T01:39:43.000Z
quantity = 6	2019-03-15T01:39:58.000Z
quantity = 2	2019-03-15T01:40:10.000Z
quantity = 0	2019-03-15T01:40:23.000Z

Figura 5.15 Notificações enviadas ao hub unitC3

- **Configurações de regras:**

Description	Parameter	Condition	Control Value	Threshold (%)
Percentagem de ocupação na prateleira	ShelfOccupation	<	0	10
Número de objetos	quantity	<	3	0

Figura 5.16 Regras configuradas para o hub unitA1

Description	Parameter	Condition	Control Value	Threshold (%)
Percentagem de ocupação da prateleira	ShelfOccupation	<	0	20
Número de objetos	quantity	<	6	0

Figura 5.17 Regras configuradas para o hub unitB2

Description	Parameter	Condition	Control Value	Threshold (%)
Percentagem de ocupação na prateleira	ShelfOccupation	>	0	30
Número d objetos	quantity	>	7	0

Figura 5.18 Regras configuradas para o hub unitC3

- **Resultados obtidos:**

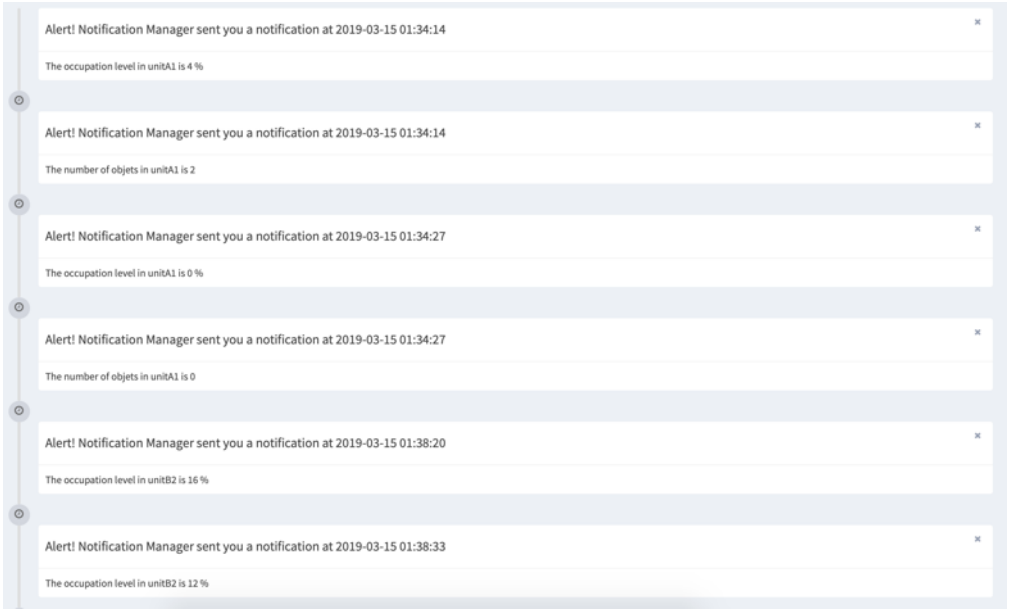


Figura 5.19 Notificações de alerta recebidas pelo utilizador parte 1

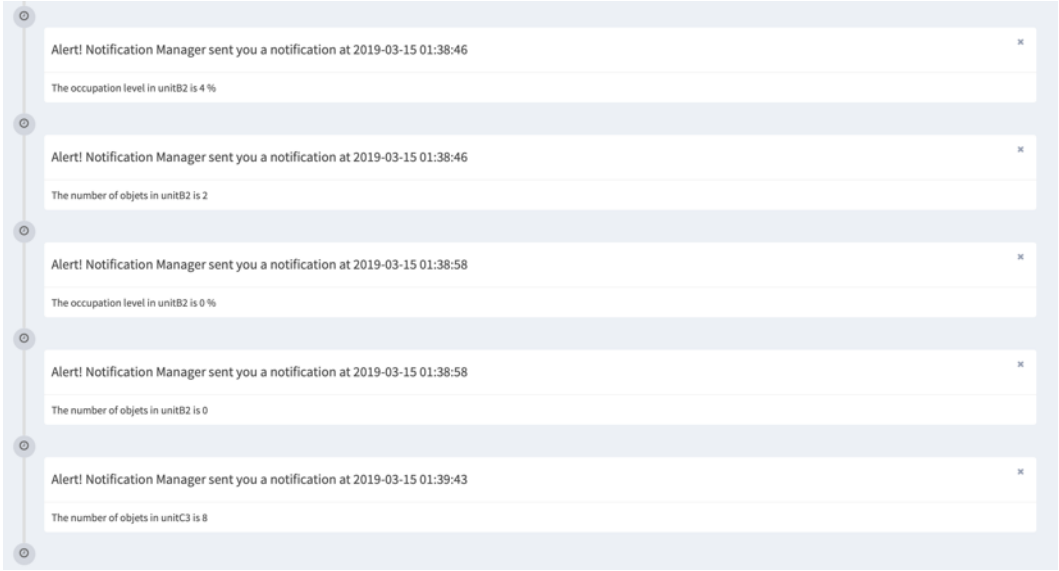


Figura 5.20 Notificações de alerta recebidas pelo utilizador parte 2

Com a análise dos resultados é possível concluir que os resultados obtidos eram os esperados. Foram recebidas cinco notificações em relação a prateleira unitA1 e seis da prateleira unitB2 num total de oito em cada uma delas. Só foi enviada uma notificação em relação à prateleira unitC3, como já era esperado, pois a regra implementada era para quando o número de objetos fosse maior que sete o que só ocorre uma vez e para uma percentagem maior que trinta porcento, algo que não aconteceu.

5.2.4 Conclusões

Dentro dos testes realizados o resultado esperado foi sempre obtido. Esse resultado é um indicador do bom funcionamento do sistema funcionamento do sistema.

6 CONCLUSÕES E TRABALHO FUTURO

No presente e último capítulo desta dissertação são apresentadas algumas considerações finais relativas ao projeto desenvolvido. Por fim também se referiu algumas ideias para possíveis trabalhos futuros.

6.1 Considerações finais

Com o aparecimento de consumidores cada vez mais digitais, é necessária uma adaptação por parte dos grandes supermercados para fortalecer a relação e melhorar a experiência. Neste contexto é possível concluir que o número de soluções implementadas é ainda baixo.

Assim sendo, sugeriu-se a implementação de um sistema inteligente com capacidade de monitorizar prateleiras em superfícies comerciais, através da análise e deteção de eventos. A solução proposta na presente dissertação foi focada principalmente na vertente de *software*, em *cloud*, tentando ao máximo proporcionar ao utilizador uma experiência de interação com o protótipo desenvolvido. Este sistema é constituído por um motor de validação de regras, denominado Notification Manager, e um módulo de análise de dados.

O primeiro passo na implementação deste sistema foi garantir a comunicação em ambiente cloud. Esta comunicação foi possível com a utilização do protocolo MQTT, que é definido por ser um protoclo simples e muito usado em sistemas IoT. Através do protocolo de comunicação MQTT os dados gerados pela parte física do projeto alcançavam os módulos na cloud. A preparação dos componentes para a recepçao dos dados foi também um passo importante pois, os dados tinham de vir em um esquema definido pelo projeto num todo. Abordando ainda o tema da comunicação, a velocidade a que os dados são envidados e processados é um requisito importante do sistema. Para alcançar esse objetivo recorreu-se ao protocolo de comunicação por Websockets. Este protocolo enquadrou-se no projeto devido a sua capacidade de comunicação em tempo real permitindo a apresentação de dados em tempo real ao utilizador.

De seguida foram separadas as bases de dados dos componentes e colocadas em um único componente que é o repositório. Este passo proporcionou maior escalabilidade e interoperabilidade ao projeto uma vez que qualquer componente poderia aceder a base de dados.

Em último lugar foram criadas regras mais específicas ao cenário proposto enquadrado. Essas regras mais específicas permitem abordar melhor os problemas mais identificados em uma superfície comercial. Problemas como a avalicação da quantidade de produtos na prateleira e quando precisam de ser repostos e também a venda. Isto é, que produtos estão a produzir maior rendimento e como melhorar essa situação. Para complementar essas regras foi também implementado um módulo de análise de dados em que é possível presenciar através de gráficos e componentes visuais o estado da prateleira. Esse componente deve-se mais a análise descritiva, informando acerca dos dados relativamente ao presente.

Por fim, foram realizados testes funcionais para assegurar que o sistema estaria a um nível cumpria os objetivos a que se propôs. Ao analisar os resultados obtidos na implementação deste sistema é possível rever elevados padrões de qualidade e eficiência.

6.2 Contribuções técnicas e científicas

As tecnologias com base em sensores, *middleware* e *cloud* que englobam este projeto, têm como principal objetivo a criação de um protótipo que possa vir a participar de uma aplicação por parte das grandes superfícies comerciais com:

- Automatização do controlo de inventário;
- A geração de inúmeros dados que vão facilitar a monitorização, em tempo real, dos produtos colocados nos expositores;
- A análise de dados em ambiente *cloud*;
- Um sistema *middleware* (como parte dessa plataforma) implementado seguindo o paradigma *fog*;

Além das contribuições já referidas, este trabalho apresenta ainda mais algumas contribuições importantes. Começando pelo estado de arte, onde foram realizadas análises de sistemas de processamento de eventos complexos e paradigmas IoT. Foram também implementados métodos que permitem a interação entre os sensores e o utilizador. Esses métodos englobam tanto regras para ajudar a monitorização eventos relevantes como configurações relativas ao comportamento do sensor. Em relação ao sistema proposto não foi elaborado um artigo científico.

6.3 Trabalho Futuro

Primeiramete seria interessante testar o sistema num ambiente real para obter um maior número de dados.

Fica também por implementar modelos preditivos em relação aos dados. Esses modelos trariam imensas vantagens através identificação de atividades fraudulentas, redução de prejuízos com melhor alocação de recursos, gerência de demanda e oferta com a captação de tendências comportamentais dos clientes. Esses modelos apresentariam também vantagens ao complementarem regras, pois regras são limitadas entre valores absolutos. O problema é que algumas variáveis podem ter variações menos consistentes e através do uso de modelos preditivos as regras poderiam aprender e adaptar-se.

REFERÊNCIAS

[1] Cushman Wakefield, "Portugal Retalho," 2018.

[2] M. Marjani *et al.*, "Big IoT Data Analytics: Architecture, Opportunities, and Open Research Challenges," *IEEE Access*, vol. 5, no. c, pp. 5247–5261, 2017.

[3] E. Welbourne *et al.*, "Building the Internet of Things Using RFID," *Internet Comput. IEEE*, vol. 13, no. 3, pp. 48–55, 2009.

[4] Y. Jadeja and K. Modi, "Cloud computing - Concepts, architecture and challenges," *2012 Int. Conf. Comput. Electron. Electr. Technol. ICCEET 2012*, pp. 877–880, 2012.

[5] A. Botta, W. De Donato, V. Persico, and A. Pescapé, "Integration of Cloud computing and Internet of Things: A survey," *Futur. Gener. Comput. Syst.*, vol. 56, pp. 684–700, 2016.

[6] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog Computing and Its Role in the Internet of Things," *Proc. first Ed. MCC Work. Mob. cloud Comput.*, pp. 13–16, 2012.

[7] L. M. Vaquero and L. Rodero-Merino, "Finding your Way in the Fog," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 5, pp. 27–32, 2014.

[8] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge Computing: Vision and Challenges," *IEEE Internet Things J.*, vol. 3, no. 5, pp. 637–646, 2016.

[9] M. Satyanarayanan, "The Promise of Edge Computing," no. June, 2015.

[10] *Distributed Systems: Concepts and Design*. 2003.

[11] M. Huhns and M. P. Singh, "Service-oriented computing: Key concepts and principles," *Proc. - Fifth Int. Conf. Commer. (COTS)-Based Softw. Syst.*, vol. 13, no. 1, pp. 1–7, 2013.

[12] B. B. M. Michelson and E. Links, "Event-Driven Architecture Overview 2011," *Architecture*, 2011.

[13] Z. Laliwala and S. Chaudhary, "Event-driven service-oriented architecture," *5th Int. Conf. Serv. Syst. Serv. Manag. - Explor. Serv. Dyn. with Sci. Innov. Technol. ICSSSM'08*, 2008.

[14] L. Lan, B. Wang, L. Zhang, R. Shi, and F. Li, "An event-driven service-oriented

architecture for the internet of things service execution," *Int. J. Online Eng.*, vol. 11, no. 2, pp. 4–8, 2015.

[15] D. G. Puranik, D. C. Feiock, and J. H. Hill, "Real-time monitoring using AJAX and WebSockets," *Proc. Int. Symp. Work. Eng. Comput. Based Syst.*, pp. 110–118, 2013.

[16] N. Naik, "Choice of effective messaging protocols for IoT systems: MQTT, CoAP, AMQP and HTTP," in *2017 IEEE International Systems Engineering Symposium (ISSE)*, 2017, pp. 1–7.

[17] M. Blount *et al.*, "Real-time analysis for intensive care: development and deployment of the artemis analytic system.," *IEEE Eng. Med. Biol. Mag.*, vol. 29, no. 2, pp. 110–8, 2010.

[18] P. D. Diamantoulakis, V. M. Kapinas, and G. K. Karagiannidis, "Big Data Analytics for Dynamic Energy Management in Smart Grids," *Big Data Res.*, vol. 2, no. 3, pp. 94–101, 2015.

[19] S. Jayasekara, S. Perera, M. Dayarathna, and S. Suhothayan, "DEBS Grand Challenge : Continuous Analytics on Geospatial Data Streams with WSO2 Complex Event Processor Categories and Subject Descriptors," pp. 277–284, 2015.

[20] A. Adi, D. Botzer, G. Nechushtai, and G. Sharon, "Complex Event Processing for Financial Services," *Serv. Comput. Work. 2006. SCW '06. IEEE*, pp. 7–12, 2006.

[21] G. Cugola and A. Margara, "Processing Flows of Information: From Data Stream to Complex Event Processing," *ACM Comput. Surv.*, vol. 44, no. 3, pp. 1–62, 2012.

[22] G. Cugola, A. Margara, M. Matteucci, and G. Tamburrelli, *Introducing uncertainty in complex event processing: model, implementation, and validation*, vol. 97, no. 2. 2014.

[23] E. Wu, Y. Diao, and S. Rizvi, "High-performance complex event processing over streams," *Proc. 2006 ACM SIGMOD Int. Conf. Manag. data - SIGMOD '06*, p. 407, 2006.

[24] WSO2, "WSO2 Complex Event Processor," *Wso2*, 2018. [Online]. Available: https://wso2.com/products/complex-event-processor/.

[25] T. Advantage, "TIBCO BusinessEvents Architect's Guide," 2016. [Online]. Available: https://docs.tibco.com/pub/businessevents-standard/5.3.0/doc/pdf/TIB_businessevents-standard_5.3_architects_guide.pdf.

[26] Esper, "Esper Reference," *Esper Core Engine*, 2012. [Online]. Available: http://www.espertech.com/download/public/EsperTech technical datasheet.pdf.

[27] F. Specific and E. Webinar, “Welcome to the FITMAN Specific Enabler Webinar on DyCEP 16,” no. June, 2015.

[28] D. Gyllstrom, E. Wu, H.-J. H. Chae, Y. Diao, P. Stahlberg, and G. Anderson, “SASE: Complex Event Processing over Streams,” *Gen. Syst.*, vol. abs/cs/061, pp. 363–374, 2006.

[29] F. Big *et al.*, “Datasheet Fujitsu Interstage Big Data Complex Event Processing Server V1 . 0.” [Online]. Available: http://www.fujitsu.com/global/documents/products/software/middleware/application-infrastructure/interstage/download/big-data/Fujitsu-Interstage-BDCEP-V1-Datasheet.pdf.

[30] “SiddhiQL Guide 3.0 - Complex Event Processor 4.0.0 - WSO2 Documentation.” [Online]. Available: https://docs.wso2.com/display/CEP400/SiddhiQL+Guide+3.0. [Accessed: 05-Sep-2018].

[31] I. Graham, *Business Rules Management and Service Oriented Architecture: A Pattern Language*. 2006.

[32] F. Hayes-Roth, “Rule-based systems,” *Commun. ACM*, vol. 28, no. 9, pp. 921–932, 1985.

[33] M. Proctor, M. Neale, P. Lin, and M. Frandsen, “Drools documentation,” *Tech. Rep.*, pp. 1–297, 2008.

[34] J. Lee, B. Bagheri, and H. A. Kao, “A Cyber-Physical Systems architecture for Industry 4.0-based manufacturing systems,” *Manuf. Lett.*, vol. 3, pp. 18–23, 2015.

[35] S. Yin and O. Kaynak, “Big Data for Modern Industry: Challenges and Trends,” *Proc. IEEE*, vol. 103, no. 2, pp. 143–146, 2015.

[36] I. A. T. Hashem, I. Yaqoob, N. B. Anuar, S. Mokhtar, A. Gani, and S. Ullah Khan, “The rise of ‘big data’ on cloud computing: Review and open research issues,” *Inf. Syst.*, vol. 47, pp. 98–115, 2015.

[37] X. Wu *et al.*, “Top 10 algorithms in data mining,” *Knowl Inf Syst*, vol. 14, pp. 1–37, 2008.

[38] L. Da Xu, W. He, and S. Li, “Internet of things in industries: A survey,” *IEEE Trans. Ind. Informatics*, vol. 10, no. 4, pp. 2233–2243, 2014.

[39] K. Shvachko, H. Kuang, S. Radia, and R. Chansler, “HDFS - The Hadoop distributed file system,” *2010 IEEE 26th Symp. Mass Storage Syst. Technol. MSST2010*, pp. 1–10, 2010.

[40] E. Laxmi Lydia and M. Ben Swarup, “Analysis of Big data through Hadoop Ecosystem

Components like Flume, MapReduce, Pig and Hive," vol. 5, no. 01, 2016.

[41] C. Douglas, J. Lowe, O. O. Malley, and B. Reed, "Apache Hadoop YARN : Yet Another Resource Negotiator."

[42] M. Zaharia, M. Chowdhury, M. J. Franklin, and S. Shenker, "Spark: Cluster Computing with Working Sets."