

Sistema de Controle de Acesso de Pessoas em um Shopping

Bruna Michelly Cordeiro e Thalia Santos de Santana

Resumo

Uma corrida pelo desenvolvimento de aplicações utilizando Tecnologias de Informação e Comunicação (TIC) tem sido presenciada dia a dia devido a pandemia causada pelo COVID-19. Visando esse cenário, o propósito deste trabalho é desenvolver um sistema que seja capaz de realizar o controle de acesso de pessoas em um Shopping Center. Esse sistema fornece o monitoramento de entrada e saída de indivíduos pelos acessos principais, além de verificar a quantidade de pessoas presentes em cada andar, com a finalidade de evitar aglomerações em pontos específicos do ambiente. O presente trabalho é dividido em quatro partes sendo: I. Núcleo funcional do sistema composto pelos componentes e suas interações; II. Coordenação, consenso e controle de concorrência; III. Tolerância a falhas e replicação; IV. Aspectos de segurança da interação entre os componentes. Como resultado, espera-se prover a criação de um sistema distribuído com vistas a prover um controle em tempo real do fluxo de pessoas em uma instalação de shopping, tendo como referência o Passeio das Águas Shopping, situado em Goiânia-GO.

Introdução

Em 31 de dezembro de 2019, a Organização Mundial da Saúde (OMS) foi alertada sobre vários casos de pneumonia na cidade de Wuhan, província de Hubei, na República Popular da China. Tratava-se de um novo tipo de coronavírus que não havia sido identificado antes em seres humanos. Os coronavírus são a segunda principal causa de resfriado comum e, até as últimas décadas, raramente causavam doenças mais graves do que o resfriado comum em seres humanos. Ao todo, sete coronavírus humanos (HCoVs) já foram identificados. O novo tipo que foi nomeado em 11 de fevereiro de 2020, recebeu o nome de SARS-CoV-2. Esse novo coronavírus é responsável por causar a doença COVID-19 [1].

Devido a pandemia é necessário que haja um controle de acesso de pessoas em locais de uso público e coletivo, a exemplo de Shoppings Centers, com a finalidade de evitar aglomerações. O sistema de controle de acesso apresentado neste trabalho é desenvolvido por meio de simulação dos dados, a exemplo do nível de visitantes no contexto de movimentação de entradas e saídas, com geração de dados aleatórios para representar as pessoas no shopping.

O trabalho está organizado do seguinte modo: a próxima seção apresenta o cenário de referência; posteriormente, há a descrição dos dispositivos; descrição dos requisitos definidos para o sistema; etapas e aspectos descritivos, em conjunto à metodologia empregada em cada uma das mesmas com informes quanto à implementação, seguido das referências bibliográficas.

Cenário

Um shopping da cidade de Goiânia foi escolhido como modelo para o projeto. Inaugurado em 2013, o Passeio das Águas Shopping é o maior centro de compras e lazer do estado de Goiás [3]. A Figura 1 apresenta uma visão geral do shopping que é composto por mais de 78 mil metros quadrados (m²) de Área Bruta Tributável (ABI). O local é administrado pela Aliansce Sonae [4].



Figura 1 - Área externa do Passeio das Águas Shopping. Fonte: [3].

Em sua estrutura, o shopping contém dois pavimentos, cerca de 283 lojas, sendo 15 âncoras e 1 hipermercado, além de 7 salas de cinema [3]. A Figura 2 apresenta a planta baixa do shopping com indicações das lojas e possíveis entradas de clientes e funcionários.



Figura 2 - Indicações das partes do shopping. Fonte: Adaptada pelas autoras (2020).

Analisando a estrutura física do shopping, é possível definir o local de instalação e a quantidade de dispositivos que serão necessários para o monitoramento da quantidade de pessoas. Para a realização de um monitoramento com mais eficiência, dois tipos de dispositivos são considerados nesta aplicação: i) Dispositivos instalados nas portas de acesso, escadas e elevadores; ii) Dispositivos instalados nas portas das lojas para verificação dos funcionários.

Para fins de desenvolvimento de um protótipo inicial, será considerada a instalação de um único dispositivo na entrada de acesso ao interior da loja, independente de seu tamanho físico. Os dispositivos instalados nas portas das lojas serão configurados com informações referentes a dimensão em m², a quantidade prevista de funcionários por hora, e a capacidade máxima da loja.

O dispositivo instalado realizará a contabilização da entrada e saída de pessoas. Essa informação estará visualmente presente em um painel que será instalado na porta de entrada da loja, com o objetivo de informar aos clientes a quantidade de pessoas no local. Dessa forma, estes poderão tomar a decisão de entrar ou não na loja naquele momento.

Nos corredores e nos locais de acesso ao shopping como portas de entrada e escadas de acesso ao primeiro andar, painéis também serão instalados com a finalidade de informar aos clientes sobre a quantidade total de pessoas por pavimento, bem como o total presente no shopping. Essas informações serão fornecidas pelos dispositivos individuais instalados em cada entrada de acesso, inclusive nas escadas e elevadores.

Ademais, também será possível visualizar as informações das lojas âncoras e o quantitativo de pessoas dentro do shopping por meio de uma página eletrônica (Figura 3). O intuito é que os clientes e demais interessados consigam verificar em tempo real, inclusive de suas residências, o quantitativo de pessoas presentes no recinto. Ademais, esta página web também oportuniza o cadastro ao administrador responsável, quanto às lojas e funcionários.



Figura 3 – Página eletrônica quanto ao quantitativo de pessoas no Shopping. Fonte: Própria (2020).

Ressalta-se que além do controle de acesso de usuários comuns, o controle de funcionários também é realizado. Desta forma, tanto as pessoas que trabalham nas lojas quanto os prestadores de serviços, e.g, entregadores de mercadorias, serão contabilizados, levando em conta o ingresso no shopping por meio de uma entrada

específica conforme indicado na Figura 2. Na entrada dos funcionários, será necessário a instalação de um sistema de biometria responsável pela autorização do trabalhador, desde que este esteja no horário previamente cadastrado. A entrada do funcionário será contabilizada no sistema de contagem de pessoas. Caso o funcionário não esteja no seu horário de trabalho, então seu ingresso deverá ocorrer por meio da entrada de clientes. No interior de cada loja também deverá ser instalado o sistema de biometria para a identificação e contagem de funcionários.

Descrição do dispositivo

O dispositivo instalado no shopping apresentará uma arquitetura padrão. Assim, será utilizado o Raspberry Pi [5] que é um computador de baixo custo e tamanho reduzido. O Raspberry Pi atuará no processamento dos dados coletados e envio das informações para outros dispositivos. Para contagem das pessoas, uma câmera com ângulo de 60° será acoplada a placa. O acesso dos funcionários ocorre em entradas específicas, dessa forma, um leitor de digital será acoplado ao hardware, apenas nas entradas indicadas na Figura 2. A Figura 4 apresenta os componentes descritos.

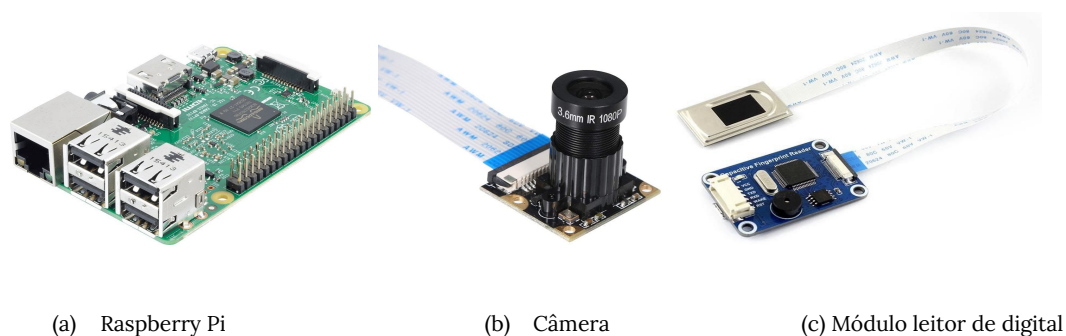


Figura 4 - Componentes utilizados na construção do dispositivo. Fonte: [5] [6] [7].

Não está no escopo do trabalho a montagem e instalação do dispositivo físico, deste modo os componentes serão meramente ilustrativos. Os dados para processamento das informações estão sendo simulados e gerados aleatoriamente, em vista de uma compreensão factível do sistema.

Descrição dos Requisitos do Sistema

Locais públicos como shoppings centers apresentam vários acessos que permitem a entrada e saída de pessoas, e normalmente possuem andares compostos por estabelecimentos comerciais, áreas de alimentação e áreas comuns onde os presentes podem se reunir. Desta forma, o sistema de controle de acesso deverá realizar o monitoramento de diferentes pontos dentro de um shopping, possuindo um conjunto definido de regras para determinado ponto monitorado. Tais regras são melhor definidas no decorrer deste trabalho.

Normalmente as pessoas que frequentam um shopping se dividem em quatro categorias: *funcionários do shopping*, *funcionários das lojas*, *prestadores de serviço* e *clientes*. Os funcionários do shopping podem ser seguranças, administradores,

profissionais da limpeza, etc. Os funcionários das lojas são as pessoas que atuam nos diferentes departamentos, em cada uma das lojas, lanchonetes, restaurantes, dentre outros. Já os prestadores de serviços são as pessoas que ocasionalmente entram no shopping, como entregadores e transportadores. E por fim, os clientes são todos os visitantes que vão ao shopping, a exemplo de atividades de lazer.

Os requisitos funcionais (RFs) referem-se a declarações de serviços que o sistema deve propiciar, com descrições de como este deve reagir a determinadas entradas bem como seu comportamento em situações específicas [2]. Portanto, os RFs atendidos pelo sistema são:

- RF01: O sistema deverá permitir ao administrador(a) do shopping cadastrar e alterar as capacidades máximas do ambiente como um todo, a taxa permitida de ocupação, além da quantidade de entradas disponíveis e quantidade de lojas;
- RF02: O sistema deverá possibilitar ao gerente da loja cadastrar e alterar os funcionários autorizados, com suas respectivas categorias e horário de entrada/saída;
- RF03: A aplicação deverá permitir, nos painéis de entrada das portas, escadas e elevadores, visualizar a quantidade de pessoas por andar e no shopping, em sua totalidade;
- RF04: A aplicação deverá permitir, nos painéis de entrada das lojas, visualizar a quantidade de pessoas presentes no respectivo ambiente;
- RF05: O sistema deverá exibir, nos painéis dos corredores do shopping, a quantidade total de pessoas presentes em cada andar;
- RF06: A aplicação deverá realizar a autenticação de funcionários e prestadores de serviço por meio de biometria, somente por entrada específica e desde que estejam no horário de trabalho cadastrado;
- RF07: O sistema deverá permitir a entrada de prestadores de serviço somente se o total de ocupantes desta categoria não tenha sido excedido;
- RF08: O sistema deverá realizar a contagem de funcionários e prestadores de serviço, visto que estes integram a contagem total de pessoas dentro do ambiente (combinada ao número de clientes);
- RF09: O sistema deverá armazenar em cada um dos dispositivos espalhados pelo shopping o status de entrada/saída;

Além das funcionalidades descritas anteriormente, o sistema deve prover a asseguuração dos seguintes requisitos não-funcionais (RNFs):

- RNF01: O sistema deverá permitir o acesso de forma segura à suas funcionalidades, realizando, para tal, a autenticação de usuários;
- RNF02: O sistema deverá possuir alta disponibilidade, de modo que os dados precisam ser mantidos de maneira distribuída, sem pontos únicos de falha;
- RNF03: O sistema deverá atender ao tempo de resposta das funções de acordo com limites estabelecidos previamente, de maneira independente à carga de uso;
- RNF04: O sistema não deverá demorar mais que 30 segundos para realizar a atualização dos painéis de fluxo de pessoas;
- RNF05: O sistema não deverá exibir quaisquer dados de cunho privativo, como dados sensíveis de funcionários, para usuários não autorizados, sendo a informação vedada somente ao administrador(a) do shopping.

Etapas e Descrição do Trabalho

O trabalho divide-se em quatro etapas que seguem um cronograma específico de entrega conforme Tabela 1. Os arquivos estão disponíveis no repositório do Github no seguinte endereço eletrônico: <<https://github.com/brunacordeiro/SCAPS>>.

Tabela 1: Cronograma das atividades do trabalho. Fonte: Própria (2020).

| Etapa | | Data Entrega (2020) |
|-------|---|---------------------|
| E1 | Núcleo funcional do sistema composto pelos componentes e suas interações | 16 de julho |
| E2 | Coordenação, consenso e controle de concorrência | 18 de agosto |
| E3 | Tolerância a falhas e replicação (confiabilidade, disponibilidade, performance) | 20 de outubro |
| E4 | Aspectos de segurança da interação entre os componentes e apresentação final | |

A primeira etapa do trabalho consiste no desenvolvimento do projeto e implementação das funcionalidades de um sistema distribuído que atenda as necessidades do cenário descrito. Assim sendo, uma arquitetura do sistema, incluindo os componentes e as interações entre eles, será projetada, implementada e testada em um ambiente distribuído real, mas com simulação dos dispositivos de IoT e dos usuários envolvidos. Nesta primeira etapa, os requisitos não-funcionais apresentados anteriormente não serão considerados.

A segunda etapa consiste em introduzir mecanismos de coordenação, consenso e controle de concorrência, de modo a garantir a consistência e coerência do estado de todos os componentes do sistema. Ou seja, elementos diferentes do sistema não poderão apresentar dados conflitantes. Nessa etapa serão descritos os mecanismos utilizados e os cenários para demonstração do estado consistente do sistema.

A terceira etapa é composta pela introdução dos mecanismos de tolerância a falhas e replicação, visando obter confiabilidade, disponibilidade, performance. Técnicas de balanceamento de carga poderão ser aplicadas para melhorar o desempenho em relação ao tempo de resposta e à vazão, bem como para aumentar a escalabilidade do sistema e a disponibilidade de componentes críticos. O sistema deverá permanecer operando mesmo mediante falhas de alguns de seus componentes, realizando-se testes de desempenho e testes de funcionalidade..

Por fim, na quarta etapa do trabalho busca-se introduzir mecanismos de autenticação e criptografia para atender ao requisito não-funcional de segurança. Usuários cadastrados deverão ser identificados de forma segura. A comunicação entre os componentes do sistema deverá ser autenticada e resistente a ataques de conteúdo.

Metodologia da Primeira Etapa

A arquitetura do sistema proposto consistirá na estruturação dos componentes de software, das propriedades desses componentes e dos relacionamentos entre estes componentes. Dessa forma, dentro do estilo de arquitetura distribuída, foi utilizado o *middleware* RabbitMQ, que ao introduzir protocolos como o AMQP 0-9-1 e/ou AMQP 1.0, bem como demais protocolos de mensagens suportados, possuem em comum a

característica de serem inerentemente distribuídos, haja vista que em grande parte conectam-se em *hosts* remotos [8].

Para ingresso no shopping, o sistema deverá verificar como dados de entrada as informações das câmeras (de modo simulado), que após processamento, proverão uma sequência de números binários representáveis como “0” para saída e “1” para entrada. Os números serão utilizados para contabilização da quantidade de pessoas dentro do recinto. Deste modo, considerando-se uma situação de implementação real, a aplicação seguiria o seguinte fluxo de atividades (Figura 5):

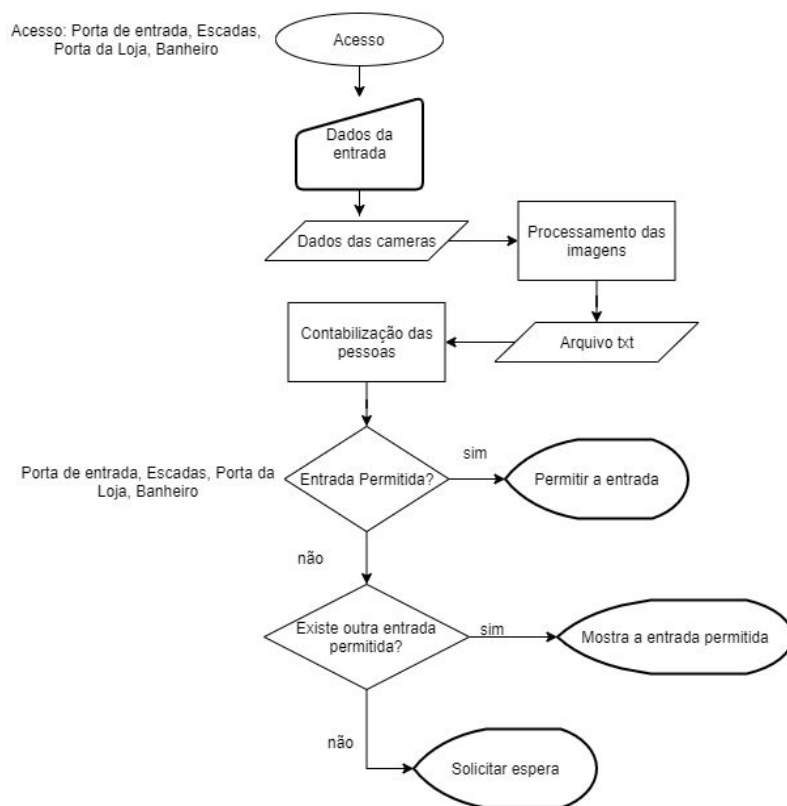


Figura 5 - Diagrama de fluxo da sequência após entrada do cliente dentro do shopping. Fonte: Própria (2020).

Neste contexto, o RabbitMQ é o *middleware* responsável para que haja comunicação entre os serviços, considerado como um dos sistemas de mensagens de código aberto mais utilizados [9]. Desenvolvido em linguagem Erlang, implementa diversos protocolos, como o HTTP e AMQP, além de possuir grande desacoplamento entre serviços. Suas mensagens por padrão caem na memória da máquina, sendo citado como extremamente rápido e poderoso, tendo como base para seu funcionamento o *Transmission Control Protocol* (TCP).

Para este projeto, o RabbitMQ trata-se do meio de campo entre o padrão *publish/subscribe* [10]. O *publisher* publica a mensagem para que for consumir e o *subscribe*, é o consumidor - capaz de ler a mensagem e utilizá-la como desejado. Entretanto, o publicador não envia sua mensagem diretamente para quem vai consumi-la, ela cai numa fila de mensagens, a qual é lida pelo consumidor.

Destaca-se que a mensagem enviada perpassa pelo *exchange* antes de chegar a fila - que pega a mensagem, processa e descobre para qual fila a mensagem vai ser enviada. Existem diversos tipos de *exchange*, como o *direct exchange* - que envia especificamente

para uma determinada fila, ou mesmo por *fanout* (tipo usado por este projeto), que é uma *exchange* que vai mandar para todas as filas que estão relacionadas [11], sendo a forma desejada para o shopping, onde as informações de entrada devem comunicar-se entre si.

Por *fanout*, conforme apresentado na Figura 6, quando manda-se a mensagem, ela cai na *exchange*, sendo replicadas pelas filas, ou seja, envia-se uma única mensagem e todas as filas vão receber.

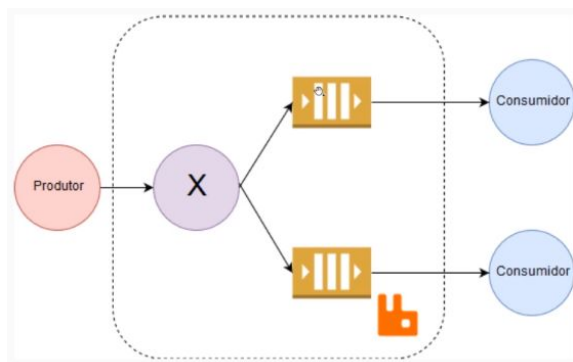


Figura 6 - Esquema representativo do envio de mensagens via *exchange* Fanout. Fonte: [8].

Desta maneira, para esta entrega, levando em conta a porta principal, a representação pelo RabbitMQ pode ser melhor vista na Figura 7. Há um *publisher* e um *consumer* (*subscriber*), que perpassa a *exchange* em uma fila de mensagens, em um único âmbito.

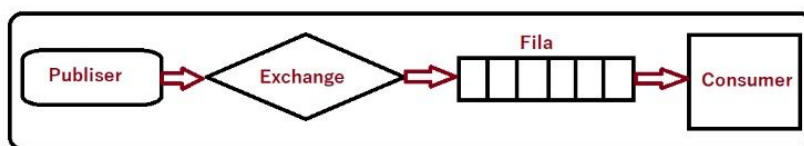


Figura 7 - Esquema publicador e consumidor para a fila da porta principal. Fonte: Própria (2020).

Para trabalhar com o RabbitMQ, foi necessário seu download e instalação, conforme tutorial em <<https://www.rabbitmq.com/download.html>>. Foi utilizada a instalação para Windows, por meio do Chocolatey, um gerenciador de pacotes para este sistema operacional.

O Chocolatey deve ser instalado previamente por meio do Power Shell do Windows, habilitando a Política de Execução do terminal para executar scripts e realizar o download de pacotes descritos em [12]. Após esta etapa, é necessário executar um comando específico de instalação apresentado em [13]. Se a instalação ocorrer corretamente, ao digitar **choco -?**, será exibida a versão do choco.

Após a instalação, é necessário de fato efetuar a instalação do RabbitMQ por meio do comando: **choco install rabbitmq**¹. Posteriormente, para verificar o envio/recebimento das mensagens, é preciso inicializar a interface do RabbitMQ, com o comando: **.\rabbitmq-plugins enable rabbitmq_management**². Seguindo esses passos será possível acessar a URL do <<http://localhost:15672/>>, como visualizado na Figura 8.

¹ como descrito em <<https://www.rabbitmq.com/install-windows.html#chocolatey>>

² como descrito em <<https://documentacao.senior.com.br/seniorxplatform/instalacao/rabbitmq.htm>>

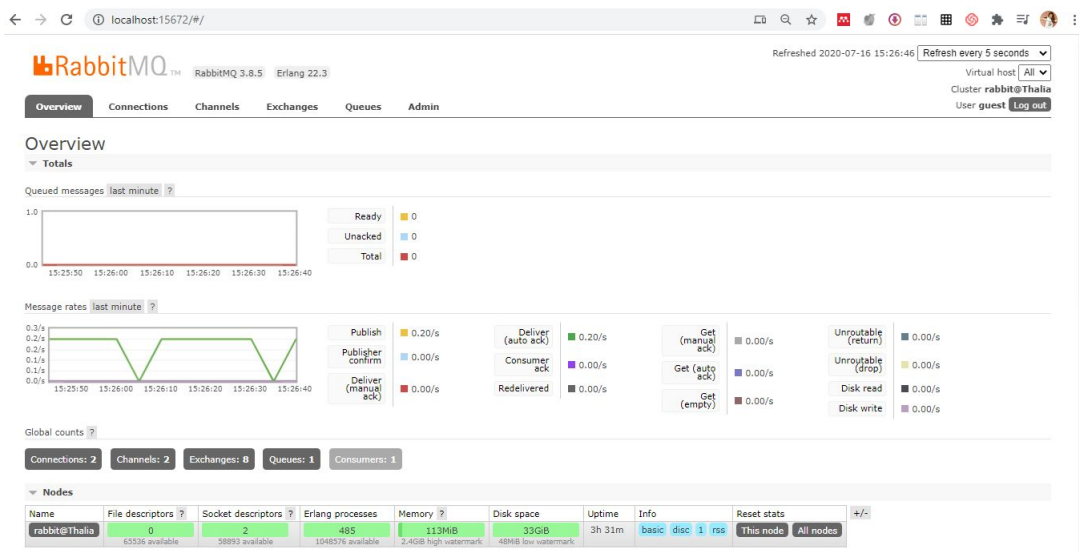


Figura 8 - Interface do RabbitMQ para visualização das mensagens. Fonte: Própria (2020).

Quanto ao código de envio/resposta das mensagens, foram criados três arquivos: *entradaPrincipal.py*, *receiver.py* e *receiver_log.py*. O primeiro refere-se a simulação da entrada principal do shopping, buscando em primeiro momento, funcionar em uma entrada do ambiente para posterior replicação. O *receiver* envia a mensagem da quantidade de pessoas entrando ou saindo, e o *receiver_log* recebe a mensagem, enquanto tudo isso é passível de monitoramento no RabbitMQ. O código fonte da primeira parte está disponível em: <<https://github.com/brunacordeiro/SCAPS/tree/master/Parte%20I>>.

Metodologia da Segunda Etapa

A segunda etapa do trabalho tem como objetivo introduzir mecanismos de coordenação, consenso e controle de concorrência, de modo a garantir a consistência e coerência do estado de todos os componentes do sistema. Assim, espera-se que elementos diferentes do sistema não tenham dados conflitantes e, portanto, dados da quantidade de pessoas presentes no shopping possam ser idênticos para quaisquer dispositivos distribuídos.

Nessa seção, os mecanismos utilizados são descritos, fazendo uso de cenários simulados para demonstrar a manutenção do estado consistente do sistema. O projeto realiza exemplificação com três (3) portas principais, três (3) escadas e três (3) elevadores. A ideia é o uso destas aplicações por intermédio de soluções em nuvem, como a Amazon Web Services (AWS), considerada uma das plataformas com maior abrangência e uso mundialmente [14].

Portanto, utilizando-se a AWS por meio do serviço *Amazon Elastic Compute Cloud* (EC2), é mantida a necessidade de instalação do RabbitMQ, seguindo os passos descritos por Mutai [15] em sistema operacional Linux (Ubuntu), distribuição padrão selecionada para criação das máquinas virtuais (VMs) deste projeto, e foram escolhidas seguindo as seguintes características:

- *Ubuntu Server 18.04 LTS (HVM), SSD Volume Type - ami-0bcc094591f354be2 (64-bit x86) / ami-0bc556e0c71e1b467 (64-bit Arm)*
- *Currently selected: t2.micro (Variable ECUs, 1 vCPUs, 2.5 GHz, Intel Xeon Family, 1 GiB memory, EBS only)*

Ademais, também foi criado um grupo de segurança específico denominado *ScapsSecurityGroup*, com regras de entrada quanto à porta padrão de conexão com o RabbitMQ (5672), para que seja possível estabelecer comunicação entre as máquinas presentes no sistema.

O código da Parte I foi replicado em quatro máquinas distintas (*publisher* Porta1, *consumer* Porta1 e *publisher* Porta2, *consumer* Porta2), a fim de verificar o funcionamento e erros quanto à consistência das informações de indivíduos no shopping, essencial para esta etapa. Foi percebido então que, considerando duas portas principais, consumindo informações das filas em diferentes máquinas por meio da AWS, haveria a necessidade de inserção de mecanismos de comunicação entre as mesmas, trocando informações entre si para promoção de estados consistentes do número de presentes no recinto.

Logo, para a Parte 2 isso foi trabalhado e é possível verificar melhorias quanto à comunicação. A exemplo disso, para as portas, foram criadas três portas principais (três *publishers*) e há somente um *consumer* para as mesmas. Cada *publisher* refere-se a uma entrada. Assim, cada porta, elevador ou escada possui um *publisher* (referenciando-se aos dispositivos Raspberry dispostos na estrutura física do shopping).

Conforme a Figura 9, retornando para a ideia das portas de entrada, os três *publishers* (um de cada porta) passarão por uma *exchange*, que encaminha para fila e todos são consumidos por apenas um *consumer* (de todas as portas). Nas demais representações de escadas e elevadores, a mesma lógica é seguida.

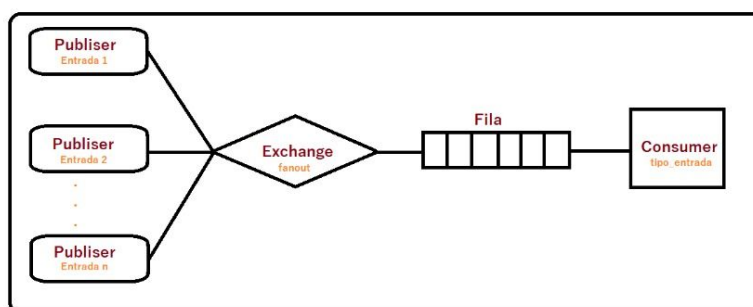


Figura 9 - Esquema publicador e consumidor para as portas principais de entrada. Fonte: Própria (2020).

Logo, foram criados três grupos de *publishers* (para os três tipos de contabilização de pessoas) e, assim, três *consumers* (derivados de cada tipo). Quanto ao consumidor, a Figura 10 exemplifica os *consumers* de portas, escadas e elevadores (que ao mesmo tempo acabam atuando tanto como *publisher* quanto *consumer*), e um *consumer* final, que deste, extrai-se dados da quantidade de pessoas dentro do shopping, bem como as informações necessárias para envio de dados visualizáveis em uma aplicação web.

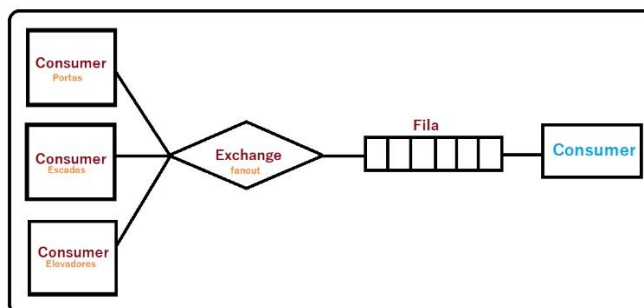
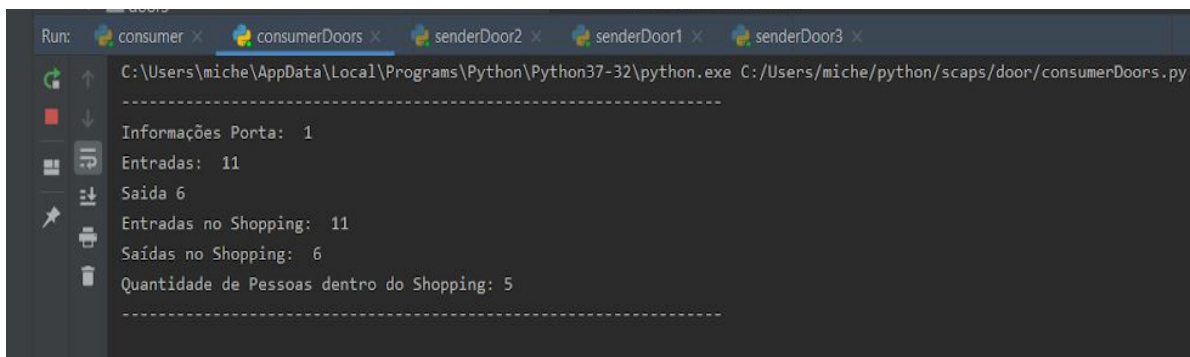


Figura 10 - Esquema entre tipos de consumers e consumidor final. Fonte: Própria (2020).

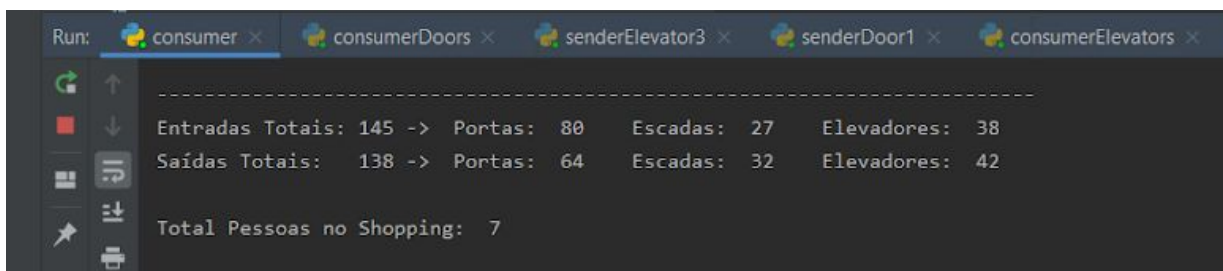
Nessa etapa, o código fonte das portas envia uma mensagem do tipo *string* para o *consumerDoors.py*. O mesmo passo é realizado para as escadas e elevadores, cada nó envia para o seu próprio consumer. A mensagem enviada é composta pela quantidade de pessoas que entraram e saíram do local monitorado. A Figura 11 apresenta a saída do *consumerDoors.py* para a porta 1, é possível notar que o resultado contém informações das Entradas e Saídas pela porta, e das entradas e saídas no shopping como todo, levando em consideração as demais portas, escadas e elevadores.



```
Run: consumer x consumerDoors x senderDoor2 x senderDoor1 x senderDoor3 x
C:\Users\miche\AppData\Local\Programs\Python\Python37-32\python.exe C:/Users/miche/python/scaps/door/consumerDoors.py
-----
Informações Porta: 1
Entradas: 11
Saída 6
Entradas no Shopping: 11
Saídas no Shopping: 6
Quantidade de Pessoas dentro do Shopping: 5
-----
```

Figura 11 - Arquivo *consumerDoors.py*. Fonte: Própria (2020).

Um controlador central foi implementado nessa etapa do trabalho, ele é responsável por prover uma única informação consistente entre todos os dados de indivíduos presentes no shopping, haja vista realizar o cálculo de entrada e saída de pessoas, considerando os cenários de portas, escadas e elevadores, por intermédio do *channel.basic_qos(prefetch_count=1)*, possibilita que o *consumer.py* (Figura 12) receba e leia uma mensagem de cada vez.



```
Run: consumer x consumerDoors x senderElevator3 x senderDoor1 x consumerElevators x
-----
Entradas Totais: 145 -> Portas: 80 Escadas: 27 Elevadores: 38
Saídas Totais: 138 -> Portas: 64 Escadas: 32 Elevadores: 42
Total Pessoas no Shopping: 7
-----
```

Figura 12 - Arquivo *consumer.py*. Fonte: Própria (2020).

Metodologia da Terceira Etapa

Esta fase representa a inserção de mecanismos de tolerância à falhas e replicação, corrigindo demais detalhes anteriores, em vistas de melhorias na vazão entre publicação e consumo de mensagens dos dispositivos presentes nas portas, elevadores e escadas. Para esta etapa, o sistema foi remodelado evitando-se a figura do coordenador central, em vista de prover de fato uma aplicação distribuída. Assim, melhorias foram implementadas quanto a coordenação do sistema, a quantidade de nós disponíveis e a aplicação.

O *middleware* utilizado no desenvolvimento do SCAPS fornece algumas funções que auxiliam na coordenação, consenso e controle de concorrência. É possível permitir que vários consumidores recebam o mesmo valor na fila de mensagens, garantindo que haja vários *subscribes* para um *publish* por exemplo. Utilizando esse recurso garante-se

que os nós que realizam o consumo das filas referentes às escadas, portas e elevadores - *consumerLadder*, *consumerDoors*, *ConsumerElevator*, respectivamente - possam receber a mesma informação.

Na documentação do RabbitMQ [16] está presente a descrição da função que realiza o roteamento da mensagem para vários consumidores. A Figura 13 apresenta um *publish P* que envia a mensagem para uma *exchange* do tipo *direct* que encaminha a mensagem para os *subscribers C₁* e *C₂*.

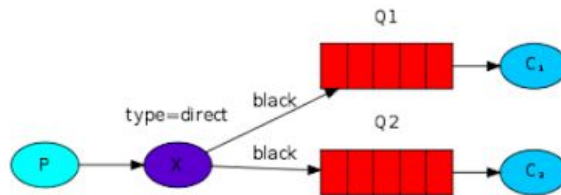


Figura 13 - Utilização de vários consumidores para um publicador. Fonte: [16].

A função utilizada para execução do roteamento da mensagem está apresentada abaixo. O *routing_key* é um parâmetro que representa uma chave de ligação. No código fonte do SCAPS o valor utilizado é: **contagem**

```
channel.queue_bind (exchange=exchange_name, queue=queue_name, routing_key='black')
```

A Figura 14 apresenta o esquema de nós que ilustra a arquitetura do SCAPS. Foram implementados 5 nós para escadas [L₁ a L₅], 3 nós para portas [P₁ a P₃] e 5 nós para elevadores [E₁ a E₅]. Esses nós realizam a leitura do ambiente que está monitorando e envia uma mensagem do tipo *string* para o consumidor (*Ladder*, *Door* ou *Elevator*) contendo informações sobre a quantidade de pessoas que entraram e saíram.

Deste modo, os nós que representam os consumidores, estão presentes na raiz da árvore (Figura 14), sendo *Ladder* - *consumerLadder.py*; *Door* - *consumerDoors.py* e *Elevator* - *consumerElevator.py*. Cada consumidor recebe a mensagem dos nós e realiza a verificação da quantidade de pessoas que estão presentes no interior do shopping.

A aplicação ilustrada na figura representa uma possível visualização do quantitativo de pessoas no shopping para o usuário final. Essa aplicação foi desenvolvida em python e se comunica com o *middleware* via protocolo AMQP para obter as informações necessárias.

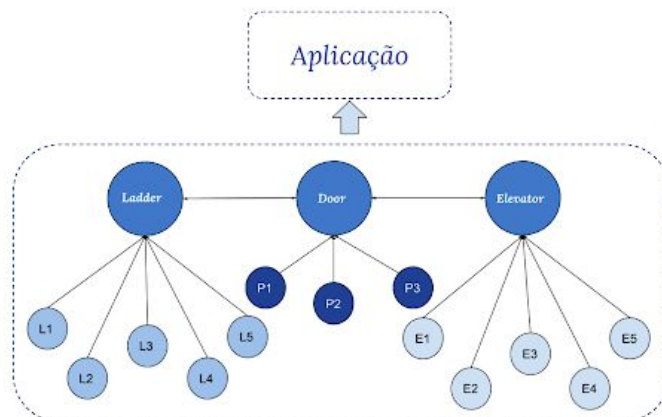


Figura 14 - Representação dos nós do sistema. Fonte: Própria (2020).

A Figura 15 apresenta com mais detalhes a arquitetura do SCAPS. É possível notar as filas criadas para cada grupo e a conexão entre os nós. Os consumidores estão conectados entre si por meio de uma fila específica, dessa forma, é possível fazer com que o consumidor da escada obtenha as informações das portas e dos elevadores, por exemplo.

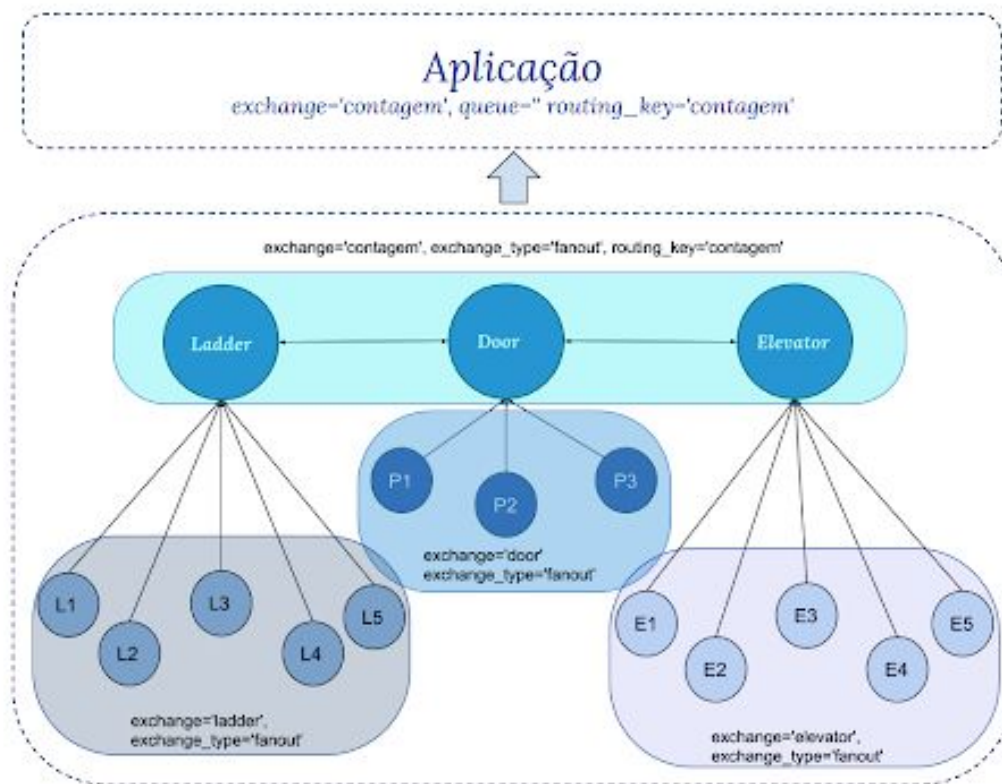


Figura 15 - Representação das filas utilizadas no sistema. Fonte: Própria (2020).

O resultado da aplicação implementada apresenta a quantidade total de clientes que entraram e saíram do shopping e dentre essa quantidade, quantos entraram pela porta, pela escada e pelo elevador. Essa informação detalhada auxilia na implementação de políticas para o acesso dos clientes quando necessário. A aplicação também mostra informações sobre a quantidade de funcionários que entraram e saíram, a implementação dos funcionários será descrita na próxima seção deste relatório.

```
-----
Entradas Clientes: 350 -> Portas: 195 Escadas: 34 Elevadores: 121
Saídas Clientes: 342 -> Portas: 194 Escadas: 33 Elevadores: 115

Entradas Funcionários: 99 Saídas Funcionários: 88

Total -> Pessoas no Shopping: 8 Funcionários no Shopping: 11
```

Figura 16 - Visualização da aplicação. Fonte: Própria (2020).

Quanto à questão de tolerância à falhas, o sistema modelado por intermédio do RabbitMQ é robusto ao passo que se algum nó que está realizando a leitura de entradas e saídas, deixar de enviar informações, por falhas ou motivos de segurança interna do

shopping, e.g bloqueio de entrada, a aplicação continua seu funcionamento da mesma forma. Ou seja, o RabbitMQ consegue perceber que os dados daquela fila não estão mais sendo publicados, e mantêm-se em execução somente com as demais.

Quando uma conexão falha, as mensagens podem estar em trânsito entre o cliente e servidor. Em tais eventos, as mensagens precisarão ser retransmitidas. Os *Acknowledgements* (Acks) permitem que o servidor e os clientes saibam quando fazer isso [17]. Os Acks podem ser usados em ambas as direções: para permitir que um consumidor indique ao servidor que recebeu e/ou processou uma entrega e para permitir que o servidor indique a mesma coisa ao consumidor. O uso de Ack garante pelo menos uma entrega. Ademais, do ponto de vista de balanceamento de cargas, ao utilizar o AMQP dentro do RabbitMQ, o balanceamento é efetuado naturalmente entre os consumidores [18]. A função representada abaixo apresenta a utilização do recurso Ack. Essa função foi utilizada em todos os códigos do SCAPS.

```
channel.basic_consume(queue=queue_contagem, on_message_callback=callback, auto_ack=True)
```

Metodologia da Quarta Etapa

A presente seção descreve como foram traçadas ações em prol de garantir requisitos de segurança em nível físico (verificação dos funcionários do shopping) e nível de sistema (validação da entrega da mensagens, definição de usuário com permissões específicas para acesso ao middleware, e validação de grupo de segurança nas máquinas virtuais).

Na implementação para verificação dos profissionais que trabalham no shopping são consideradas quatro categorias: Funcionário de Loja; Prestador de Serviço; Equipe de Segurança e Equipe de Limpeza.

Os Funcionários de Loja, vendedores ou gerentes, são vinculados às lojas do shopping. Cada loja possui um cadastro onde é informado o nome e a quantidade de funcionários. Assim que o trabalhador entra no shopping, através de entrada específica conforme Figura 2, sua impressão digital é solicitada, por meio dessa informação, é possível categorizar o funcionário e identificar a função específica.

A Figura 17 apresenta a execução do código *senderWorker.py* é possível ver a categorização dos funcionários e a loja a qual está vinculado.

```
Entrada detectada!
Categoria: Funcionário Loja
Loja: Marisa

Entrada detectada!
Categoria: Equipe de Segurança

Entrada detectada!
Categoria: Prestador de Serviço
Loja: Marisa
```

Figura 17 - resultado do código *senderWorker.py*. Fonte: Própria (2020).

Na Figura 18 é possível visualizar a execução do consumer da implementação, informações sobre a quantidade de funcionários que estão no interior da loja são apresentadas. Quando um funcionário finaliza suas atividades e deixa o estabelecimento, a quantidade é atualizada.


```
Qnt de funcionários Loja Centauro : 12
Qnt de funcionários Loja Centauro : 13
Equipe de Segurança no Shopping: 12
Polo Wear Todos os funcionários estão presentes!
Equipe de Limpeza no Shopping: 12
Equipe de Segurança no Shopping: 13
```

Figura 18 - resultado do código `consumerWorker.py`. Fonte: Própria (2020).

A segurança a nível de sistema utiliza grupos de segurança na AWS garantindo que somente as máquinas presentes no grupo estabelecem comunicação entre si. Do ponto de vista do *middleware* foram apresentadas neste documento funções que realizam a segurança no envio das mensagens, como os recursos de qualidade de serviço e Ack. Não somente em relação ao envio das mensagens, mas a autenticação no RabbitMQ que refere-se ao usuário de acesso foi atualizada e não utiliza-se o usuário padrão do sistema. Neste caso, foi criado e adicionado um novo usuário, com uma senha específica ao projeto, além de configurações de permissões, como exemplificado abaixo:

```
sudo rabbitmqctl add_user scaps 123456789
sudo rabbitmqctl set_user_tags scaps administrator
sudo rabbitmqctl set_permissions -p / scaps "." "." ".*"
```

Para trabalhos futuros, objetiva-se organizar as informações advindas da aplicação em uma página eletrônica, facilitando a visualização de indicadores de entrada e saída do shopping, como já apresentado o protótipo inicial na Figura 3.

Por fim, a apresentação em formato de *videocast* pode ser conferida em: <<https://www.youtube.com/watch?v=OE9gfdzQPvY>>.

Referências

- [1] OPAS/OMS Brasil. Folha informativa - COVID-19 (doença causada pelo novo coronavírus). 2020. Disponível em: <https://www.paho.org/bra/index.php?option=com_content&view=article&id=6101:covid19&Itemid=875>. Acesso em: 1 julho. 2020.
- [2] SOMMERVILLE, Ian. Engenharia de Software. 9. ed. São Paulo: Pearson. Prentice Hall, 2011.
- [3] Passeio das Águas Shopping. Sobre o Shopping. Disponível em: <<https://passeiodasaguasshopping.com.br/sobre-o-shopping/>>. Acesso em: 3 julho. 2020
- [4] Aliansce Sonae. Excelência e comprometimento. Disponível em: <<http://www.aliansce-sonae.com.br/>>. Acesso em: 3 julho. 2020
- [5] Raspberry Pi. What is a Raspberry Pi?. Disponível em: <<https://www.raspberrypi.org/help/what-%20is-a-raspberry-pi/>>. Acesso em: 3 julho. 2020.
- [6] Amazon. Docooler Wide Angle Camera 5M Pixel Adjustable Angle Compatible for Raspberry Pi 3 Model B B+. Disponível em: <<https://www.amazon.com/Docooler-Camera-Adjustable-Compatible-Raspberry/dp/B07K56KFX5>>. Acesso em: 07 julho. 2020.

[7] Módulo de impressão digital capacitivo Alta precisão de aquisição e módulo de reconhecimento. Disponível em: <<https://pt.dhgate.com/product/capacitive-fingerprint-module-high-precision/432120605.html>>. Acesso em: 20 de outubro de 2020.

[8] RabbitMQ. Distributed RabbitMQ. Disponível em: <<https://www.rabbitmq.com/distributed.html>>. Acesso em: 14 julho 2020.

[9] RabbitMQ. Quorum queues. Disponível em: <<https://www.rabbitmq.com/quorum-queues.html>>. Acesso em: 14 julho 2020.

[10] RabbitMQ. Publish/Subscribe (using the Pika Python client). Disponível em: <<https://www.rabbitmq.com/tutorials/tutorial-three-python.html>>. Acesso em: 14 julho 2020.

[11] Gist Github (Renato Costa). RabbitMQ com Python. Disponível em: <<https://gist.github.com/renatoapcosta/2a4b6c7a5933edf09e9226e1f1ca989>>. Acesso em: 16 julho 2020.

[12] Conhecendo o Chocolatey. Disponível em: <<https://hcode.com.br/blog/conhecendo-chocolatey-o-gerenciador-de-pacotes-do-windows>>. Acesso em: 06 agost. 2020.

[13] Install Chocolatey. Disponível em: <<https://chocolatey.org/install>>. Acesso em: 06 agost. 2020.

[14] Amazon Web Services. Computação em nuvem com a AWS. Disponível em: <https://aws.amazon.com/pt/what-is-aws/?nc1=f_cc>. Acesso em: 18 agost. 2020.

[15] Mutai, Josphat. Computer for Geeks. How To Install Latest RabbitMQ Server on Ubuntu 20.04 | 18.04 LTS. Disponível em: <<https://computingforgeeks.com/how-to-install-latest-rabbitmq-server-on-ubuntu-linux/>>. Acesso em: 06 agost. 2020.

[16] RabbitMQ. Routing. Disponível em: <<https://www.rabbitmq.com/tutorials/tutorial-four-python.html>>. Acesso em: 19 out. 2020.

[17] RabbitMQ. Reliability Guide. Disponível em: <<https://www.rabbitmq.com/reliability.html>>. Acesso em: 19 out. 2020.

[18] RabbitMQ. AMQP Concepts. Disponível em: <<https://www.rabbitmq.com/tutorials/amqp-concepts.html>>. Acesso em: 16 out. 2020.