ROBOT PI: SISTEMA DE CONTROLE PARA OPERAÇÃO DE DRONES TERRESTRES[1]

Thiago Sordi[2], Ricardo Augusto Manfredini[3]

Resumo

A Quarta Revolução Industrial, conhecida por Indústria 4.0, surge como uma evolução de suas antecessoras, introduzindo medidas contemporâneas para resolver problemas antigos. Nesse contexto, muito se fala em automatizar empresas, e portanto, aumentar a produtividade. Para isso, foi pensado no desenvolvimento de um drone terrestre para solucionar o gasto de tempo de funcionários no transporte de materiais dentro do ambiente de trabalho. Não bastando apenas o drone, este trabalho propôs também a montagem de um sistema para controlar-se e administrar-se o mesmo, como um aplicativo móvel para o envio de comandos ao drone, e o desenvolvimento de um *WebService* com diversas funcionalidades além das requisições REST, como por exemplo, um servidor com protocolo de comunicação TCP/IP. Todas as partes foram implementadas utilizando tecnologias atuais e conhecidas, como o uso do Android, Raspberry Pi, Python, entre outras. No fim de todo processo, encontram-se em funcionamento o aplicativo e o *WebService*, faltando apenas a integração do drone ao resto do sistema. Por tais razões, e considerando os testes realizados, o trabalho conseguiu se inserir dentro da Indústria 4.0, e os resultados obtidos acabaram sendo superiores ao esperado.

Palavras-chave: Indústria 4.0. Aplicativo. Drone. Raspberry Pi.

1 Introdução

Com a aparição do ser humano na Terra, da forma como conhecemos, surgiu com ele o trabalho, no princípio, ele se dava como uma forma de subsistência para nos mantermos vivos e também, como uma forma de organizar nossas comunidades, esse ideal acabou se mantendo até os dias atuais. Porém, o ser humano não passou a depender do trabalho para seu próprio sustento somente, mas o utilizou para explorar e inovar tecnologias, passando por grandes marcos, como foi o caso das diferentes Revoluções Industriais, e nisso, sempre buscou trazer formas de otimizar a produtividade de seus locais de trabalho, ou seja, suas empresas. Basicamente, o ser humano utilizou o trabalho para facilitar o trabalho, e um dos

[1] Trabalho apresentado como requisito parcial para obtenção do diploma do Curso Técnico em Informática Integrado ao Ensino Médio do ano de 2018.

[2] Aluno do Curso Técnico em Informática Integrado ao Ensino Médio do IFRS Campus Farroupilha (thiagosordi10@gmail.com).

[3] Professor orientador do Trabalho de Conclusão de Curso (ricardo.manfredini@farroupilha.ifrs.edu.br).

casos mais famosos pode ser observado no Fordismo, como descrito por Thomaz Wood (1992, p. 4).

> O conceito-chave da produção em massa não é a idéia de linha contínua, como muitos pensam, mas a completa e consistente intercambiabilidade de partes, e a simplicidade de montagem.[...] As mudanças implantadas permitiram reduzir o esforço humano na montagem, aumentar a produtividade e diminuir os custos proporcionalmente à elevação do volume produzido.[...] Em contraste com o que ocorria no sistema de produção manual, o trabalhador da linha de montagem tinha apenas uma tarefa. Ele não comandava componentes, não preparava ou reparava equipamentos, nem inspecionava a qualidade.[...]

Obviamente, não foi um meio de produção que definiu um padrão de funcionamento de empresas. No Fordismo, por exemplo, produzia-se muito, estocava-se muito, e lucrava-se pouco, por causa disso, vieram outros para tentar corrigir os problemas dos antecessores, como foi o caso do Toyotismo. Mas o que importa foi a ideia que surgiu ali e que precisava ser mantida, o aproveitamento do tempo e o uso do funcionário, algo muito importante para a produtividade da empresa, porém, que ainda não se estabeleceu uma regra concreta para isso. Pode-se ver essa situação em dois casos reais, empresas que buscam dar mais liberdade em ambiente de trabalho e sem tanta pressão, buscando a melhor performance do empregado, em contraste com outras que zelam pelo trabalho integral como forma de aproveitamento.

No entanto, independentemente da forma com que a empresa se comporta em relação ao uso do funcionário, o mundo atual já se provou uma máquina de consumo. De acordo com a Brasscom (Associação Brasileira das Empresas de Tecnologia da Informação e Comunicação), em 2017, no Brasil foi faturado R$467,8 bilhões no mercado de tecnologia (MELO, 2018). Dessa forma, vale ressaltar, que o cliente sempre buscará o que for mais rentável para ele, e a empresa, da mesma forma, tentará atrair o cliente competindo no preço, na qualidade ou em ambos, mas para que não se tenha mais despesas do que lucro, sempre busca-se otimizar em todas áreas possíveis de forma a reduzir custos, e novamente, mostra-se relevante o uso eficaz do funcionário.

Pensando nisso, e utilizando-se de medidas contemporâneas para um modo de produção em que todos se beneficiem, buscou-se utilizar como o conceito central deste trabalho, a Indústria 4.0, onde faz-se com que máquinas cumpram trabalhos que antes podiam ser feitos por humanos. Nesse trabalho foi pensando como objetivo principal a montagem de

um sistema para drones terrestres capazes de se moverem dentro de um ambiente e transportar materiais para entregas em estações, utilizando-se tecnologias relativamente comuns e materiais de baixo custo. Dessa forma, o tempo gasto pelo funcionário com uma tarefa ordinária e simples diminuiria, permitindo seu foco em uma tarefa mais importante. Ainda assim, o sistema não é totalmente auto-suficiente, é necessário uma intervenção humana, a qual tem a função de comandar o drone através de um aplicativo *mobile* intuitivo desenvolvido para Android, ao passo que esse se comunica ao intermediário, *WebService*, onde ocorrem todos os controles do sistema e o "diálogo" com o próprio drone.

2 Referencial Teórico

Indústria 4.0 é um conceito que tem estado em alta nos últimos anos, ainda mais quando pondera-se mediante à revolução tecnológica no contexto da contemporaneidade. Ela surge como uma demanda empresarial para apropriação de medidas atuais para problemas que já existem há um bom tempo, como explicado por Cristiano Bertulucci Silveira e Guilherme Cano Lopes ([2016]).

> Seu fundamento básico implica que conectando máquinas, sistemas e ativos, as empresas poderão criar redes inteligentes ao longo de toda a cadeia de valor que podem controlar os módulos da produção de forma autônoma. Ou seja, as fábricas inteligentes terão a capacidade e autonomia para agendar manutenções, prever falhas nos processos e se adaptar aos requisitos e mudanças não planejadas na produção.

Um dos problemas enfrentados pelas indústrias modernas está relacionado ao funcionário, esse que pode ser submetido a tarefa perigosas, materiais perigosos, fadiga, entre outros, e com isso, acaba por desencadear acidentes de trabalho e prejuízos a empresa. "Descuido, falta de equipamentos de segurança e até exaustão provocam 700 mil acidentes de trabalho por ano em todo o país" (SOUZA, 2017). Em todos os casos de acidentes, um funcionário robô tenderia a ter uma melhor performance de trabalho, principalmente em casos que envolvem alturas elevadas, pois dessa forma, além de ser programado para exclusivamente realizar qualquer que seja sua operação sem que haja medo e cansaço envolvidos, qualquer inconveniência que viesse a ser sofrida poderia ser resolvida por uma substituição do funcionário robô sem danos maiores do que danos materiais.

Outro problema seria a questão de tempo e esforço, numa Indústria 4.0 temos um funcionário mais focado em sua tarefa, sem desperdiçar sua energia e tempo com atividades que um robô poderia facilmente solucionar. O Instituto Global McKinsey aponta que nos próximos anos, uma boa parte de todas horas trabalhadas no mundo podem ser automatizadas (ONU BRASIL, 2018). Isso abre margem para um sistema de produção mais eficaz e menos desgastante para todos envolvidos.

Solucionando esses problemas, e isso somado a redução de custos que a empresa obterá com certos cortes, acaba fazendo esse novo modelo industrial atrativo para as empresas, isso pode ser observado na pesquisa feita pela Federação das Indústrias do Estado de São Paulo (FIESP, 2018), das 277 empresas que estiveram presente, 90% alegou que a Indústria 4.0 ajudará com a produtividade e que "é uma oportunidade ao invés de um risco". Tal vantagem empresarial está segundo Tiago Fonseca Albuquerque Cavalcanti Sigahi e Bárbara Cristina de Andrade (2017, p. 2).

> De acordo com relatório publicado pela Confederação Nacional da Indústria (2016, p. 17), estima-se que, até 2025, os processos relacionados à Indústria 4.0 poderão reduzir custos de manutenção de equipamentos em até 40%, reduzir o consumo de energia em até 20% e aumentar a eficiência do trabalho em até 25%, podendo impactar o PIB brasileiro em aproximadamente US$ 39 bilhões até 2030.

Contudo, esse novo modelo de estrutura de trabalho acaba não se sustentando sozinho, ele se apoia em tendências tecnológicas amplamente conhecidas atualmente: como a Internet das Coisas (IoT), onde se estabelece uma rede de conexões via internet entre os mais diversos dispositivos, desde smartphones até sistemas embarcados (geladeiras, *smartwatches*, entre outros), e por fim, permite uma troca de informações entre os mesmos. Além dela, a tecnologia de *Big Data,* que se refere ao processo de coleta, processamento e estudo de uma grande quantia de dados, também acaba sendo um pilar, já que utilizando os dispositivos ligados pela IoT é possível capturar e armazenar os mais diversos tipos de dados, com isso pode-se calcular resultados desses dados, assim como premeditar situações futuras. E também, tem-se a segurança virtual, que já é um assunto conhecido por muitos em tempos de internet, e por isso deve ser tratado com a devida importância, e como uma prioridade num sistema onde se espera que vários dispositivos se conectem e não sejam comprometidos, ou até mesmo vulnerabilizando a própria empresa.

Como mencionado, para a aplicação da Indústria 4.0 são utilizadas medidas atuais para solucionar ou otimizar situações industriais, e nisso pode-se pensar em ferramentas utilizadas para lazer, mas que podem vir a serem usadas dentro de uma indústria, como é o caso de um drone, por exemplo. Apesar de drones serem veículos muito conhecidos hoje em dia, ainda não são de total acesso à população em geral, muito se deve ao elevado preço, mas para suprir isso, existem programas DIY (*Do It Yourself*) que tentam utilizar materiais baratos e *open-source* para criação de drones de uso pessoal.

> Nascidos como instrumentos de defesa e ataque, os drones agora ganham uma pacífica e nobre função: ajudar no aumento da oferta mundial de alimentos e demais produtos originários do campo. Há pelo menos um ano estes aparelhos monitoram extensas culturas industriais como eucalipto e cana-de-açúcar, registrando imagens de plantio, cultivo e corte. Nos últimos meses, passaram a ser vistos em fazendas, sobrevoando áreas de soja, milho e algodão (MESQUITA, 2014, p. 21).

Apesar do seu uso majoritário em áreas rurais, drones acabam se encaixando perfeitamente na Indústria 4.0, muito por causa de terem a capacidade de utilizar os pilares da mesma. Drones podem ter sistemas embarcados que permita-lhes conexão via internet, assim como a capacidade de gerar dados para posterior análise. No caso do presente trabalho, o drone não será um dispositivo com capacidade de locomoção aérea, como a maioria, ele será limitado a trafegar pelo chão e se orientando por uma linha. O fato da sua presença ser em solo previne que certo acidentes possam vir a acontecer, como a queda do mesmo, ou até mesmo colisões com objetos ou pessoas.

Para a aplicação de um drone na Indústria 4.0, deve-se utilizar tecnologias capazes de torná-lo um dispositivo “independente”, ou seja, ele deve se tornar um computador, onde possa ser programado e, a partir disso, fazer sua tarefa sem dependência de terceiros. Visando a busca por tecnologias baratas para solucionar-se o problema, há o Raspberry Pi representado na Figura 1, que é uma tecnologia relativamente nova no mercado, conhecido formalmente pelo nome de *Single-Board Computer* (SBC), ou seja, um computador em uma placa, e como observado na imagem, pode-se notar a presença de componentes que são essenciais num computador, portas USB, saídas HDMI, processador, e outros. Como todo computador, o Raspberry Pi também precisa de um sistema operacional, existindo diversos para várias aplicações diferentes, mas o mais comum é o Raspbian, baseado no sistema Debian (distribuição Linux), com diferentes versões, e feita especialmente para esse computador.

O Raspberry Pi consegue ter uma série de vantagens, a começar pelo baixo custo quando comparado a um computador convencional, pois pode ser encontrado por US$35 em alguns locais, em seguida, suas dimensões, que permitem seu uso em diversos projetos onde precisa-se de um computador. Por essas razões, ele tem sido uma das principais tecnologias utilizada na IoT e Indústria 4.0.

> [...] ficou claro que o Raspberry Pi é uma alternativa de automação viável economicamente para simples e não críticos processos industriais, podendo ser programado e ajustado de diversas formas, visando a redução de custos da produção. Considerando que o ramo da automação industrial brasileira faturou R$4,508 bilhões em 2015 o mercado para pequenas automações pode ser potencialmente grande (ANDRADE, SOMA, ARASHIRO, 2016, p. 5).

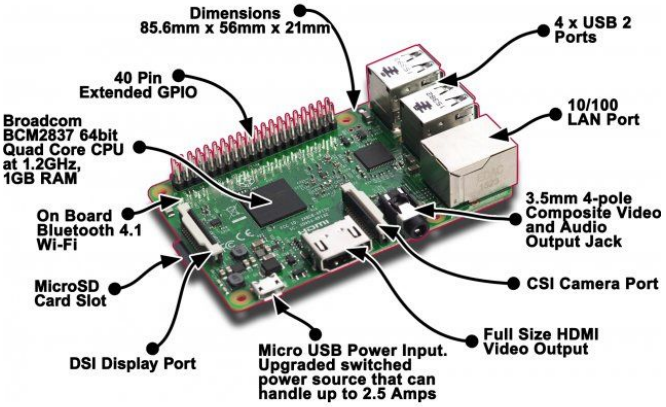


Figura 1 – Raspberry Pi 3 Model B

Fonte: Play-Asia[4]

[4] Disponível em: https://www.play-asia.com/raspberry-pi-3-model-b/13/709v93

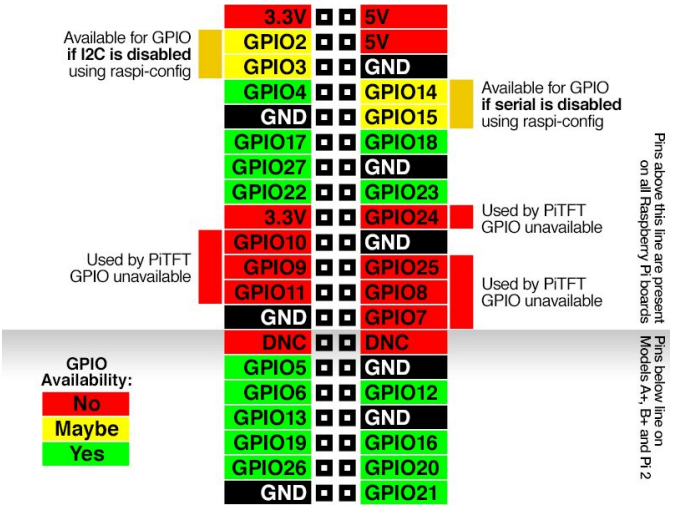


Figura 2 – Pinagem GPIO do Raspberry Pi 3 Model B

Fonte: Python Programming Tutorials[5]

Ao longo dos anos foram sendo lançadas diferentes versões do Raspberry Pi, variando memória RAM, processador, e módulos (como o Wi-Fi). Optando pela versão mais recente até o momento de escrita deste trabalho, teremos o Raspberry Pi 3 Model B, essa simples placa atinge a marca de 1 GB de RAM, possuindo CPU Quad-core 1,2 GHz, e com os módulos Wi-FI e Bluetooth já integrados. Mas houveram coisas que nunca mudaram, por exemplo, a presença de uma memória externa para salvar dados (MicroSD), e a disponibilidade dos pinos GPIO (*General Purpose Input/Output*), como pode ser observado na Figura 1. É por causa dessa pinagem que o Raspberry Pi tem sua utilidade em projetos atuais, pois ela permite a conexão a motores, sensores, entre outros. Para isso se utiliza de um esquema de pinagem como mostrado na Figura 2, com pinos de alimentação (5V, GND, 3.3V) e pinos de entrada e saída de informação (GPIO).

Concomitante ao Raspberry Pi, ainda pensando em medidas atuais para serem implementadas na Indústria 4.0, existe uma tecnologia quase universal e de fácil acesso, os smartphones. “O número de usuários únicos de telefone celular chegou a cinco bilhões no mundo em 2017, segundo dados do relatório da GSMA, associação que reúne a indústria e organiza o Mobile World Congress (MWC)” (O GLOBO, 2018). E existe um motivo para esse número de usuários ser tão alto, tais aparelhos, além de fazerem operações básicas para um celular (SMS, ligações, etc.), têm a capacidade para suportar os mais diversos aplicativos, e em alguns casos chegam a se equiparar a alguns modelos de notebooks ou desktops. Mas

[5] Disponível em: https://pythonprogramming.net/gpio-raspberry-pi-tutorials/

um smartphone por si só não possui muita utilidade sem a presença de um sistema operacional, e para suprir isso há o Android, um sistema operacional que está em alta há alguns anos e que promete não deixar tal posição por enquanto.

A Google foi a principal empresa responsável pelo desenvolvimento do sistema Android como conhecemos, mas para isso se apoiou num consórcio, conhecido como Open Handset Alliance, o qual incluía a própria Google, e algumas outras como HTC, Sony, Samsung, Qualcomm, etc. A ideia era conseguir tornar o Android o sistema operacional mais utilizado no mundo, com o principal diferencial de ser *open-source*. Ao que tudo indica, ele conseguiu atingir o que buscava, segundo StatCounter, em 2017, o Windows, sistema operacional mais utilizado até então, estava em 37,91% dos aparelhos em que opera, e o Android em 37,93% (G1, 2017). Tal fator anda em conjunto ao fato de que adquirir um computador de bolso é muito mais fácil do que um computador de mesa.

Além de atingir usuários em massa, consequentemente, o Android consegue atrair profissionais para trabalhar nele facilmente, sendo uma das áreas da tecnologia com maior demanda por profissionais. Para lidar com isso, a Google já está preparada, disponibilizando gratuitamente um *Integrated Development Environment* (IDE) ou ambiente de desenvolvimento integrado, conhecido como Android Studio. Para aprender a utilizar a IDE e criar seus próprios projetos, existem diversos tutoriais na documentação oficial online do Android e no seu kit de desenvolvimento de Software (SDK).

3 Metodologia

Este trabalho foi pensado e elaborado para se estabelecer através de sua separação em três ambientes que, após finalizados, foram integrados. Essas áreas são: (I) A montagem e programação do drone utilizando Raspberry Pi, para andar entre estações realizando sua principal tarefa, a entrega; (II) O aplicativo mobile para Android e seu uso por funcionários em geral que queiram operar o drone; (III) O *WebService* em Java que recebe as requisições do Android e as processa, mas também com a capacidade de realizar operações em paralelo para garantir a praticidade do sistema, além de ter integrado um servidor com protocolo TCP/IP para a comunicação com os drones, e uma aplicação web simples para controle do sistema em geral.

No princípio foram elaboradas algumas definições para o trabalho, como os casos de usos e como se daria o fluxo de funcionamento integrando os três ambientes. Para realizar essas definições do projeto utilizou-se o sistema web LucidChart[6] que oferece diversas ferramentas para prototipação e elaboração, mesmo em sua versão gratuita, neste trabalho uma de suas aplicações foi para os diagramas de casos de uso. E também o uso do software *open-source* Fritzing[7] para montagem de esquemas referentes a parte física do drone.

3.1 Metodologia do *WebService*

Dentre uma das principais tecnologias desse trabalho está a linguagem de programação orientada a objetos Java, ela que consegue ser tanto acessível quanto fácil de utilizar no desenvolvimento, pode nos fornecer um amplo suporte da comunidade, e oferecer diversas bibliotecas em sua documentação. Java vem a ser muito útil quando trata-se de *server-side,* pois, quando cria-se um sistema web, não se busca a dependência mediante sistemas operacionais. Com essa versatilidade que se tem, buscou-se explorar seu uso, utilizando-a em dois dos três ambientes desse trabalho, sendo um deles o *WebService*.

Para o desenvolvimento do ambiente *WebService* ser feito em Java utilizou-se uma IDE *open-source*, conhecida como Eclipse[8], capaz de interpretar diversas linguagens através de plugins. Optou-se pela utilização desta justamente pelo fato da ampla gama de plugins para cada necessidade que o desenvolvedor possa ter.

Em conjunto com o Eclipse, utilizou-se também o Spring Boot[9], disponibilizado pela Spring, ele tem como principal funcionalidade facilitar o desenvolvimento sem preocupar-se com configurações da aplicação, com isso, basta inserir as dependências que serão necessárias no projeto, como web, segurança, banco de dados, entre outros, e ele fará o resto do trabalho, essas dependências são conhecidas também como *starters*. Aproveitando também o suporte dado pela Spring, além de sua excelente documentação, tem-se o Spring Initiliazr, onde pode-se montar a base de um projeto Spring e baixá-lo. No caso deste trabalho, foi inserido para gerar um projeto Maven para Java 1.8, com a versão 1.3.6.RELEASE do Spring Boot, a versão escolhida para o Spring boot, será a versão *parent* para os *starters* da aplicação. Alguns desses *starters* foram WEB, JPA.

---

[6] Lucidchart 2018 - https://www.lucidchart.com/
[7] Fritzing versão 0.9.3 - http://fritzing.org/download/
[8] Eclipse versão Photon - https://www.eclipse.org/
[9] Spring Boot versão 1.3.6.RELEASE - https://start.spring.io/

Por ser um projeto Maven (versão 4.0.0), há um controle sobre todas as dependências, essas que ficam organizadas dentro do arquivo *pom.xml* (*Project Object Model*), e caso alguma dependência venha a ser adicionada posteriormente ao projeto, deve-se inserir nesse arquivo, e ele fará o download do *.jar* para o *classpath* do projeto, como foi o caso do JDBC (*Java Database Connectivity*) para a conexão da aplicação Java com o banco de dados MySQL.

O *starter* WEB é o que oferece ao projeto a possibilidade de criar uma aplicação RESTful com Spring MVC. O RESTful diz respeito a uma aplicação com o princípio REST, ou seja, com a capacidade de receber requisições (GET, POST, PUT, DELETE, etc.) de variados dispositivos via internet, e tratar essas requisições, e em alguns casos, retornar um resultado. Normalmente esse resultado é retornado no formato JSON (*JavaScript Object Notation*), e graças a isso, juntamente ao Jackson (processador de JSON que facilita a conversão de objetos em JSON e vice-versa), a aplicação Android deste trabalho pôde manipular objetos recebidos e enviados ao *WebService*. Já o Spring MVC fornece uma praticidade nessa operação REST e em muitas outras, começando pelo fato de possuir o padrão de arquitetura MVC (*ModelViewController*), isso obriga o projeto a manter um certo nível de organização e fluxo. Além disso o Spring MVC funciona com um sistema de anotações, estas que serão abordadas na Seção 4.1 junto a exemplos práticos.

Já o *starter* JPA permite o uso do Spring Data JPA (*Java Persistence API*) com Hibernate, com isso, não é preciso se preocupar com a conversão dos objetos da aplicação em *queries*, ou seja, a implementação de DAOs (*Data Access Object*), pois pode-se ter um alto custo de manutenção toda vez que há uma alteração a ser feita. Essa função será feita pelo JPA, o Mapeador Objeto-Relacional. Neste projeto utilizou-se como *framework* de persistência de dados, o Hibernate, uma ferramenta para a tarefa de mapeamento de objetos, ele implementa o JPA, e se orienta através de uma anotação *@Entity* em cada classe. Dessa forma, ele fará o mapeamento e a persistência dos dados, toda classe anotada virará uma tabela no banco, por isso deve-se implementar um atributo que será chave primária e anotá-lo com *@Id*, já que todos os atributos viram colunas de suas tabelas e todas tabelas precisam de uma chave primária.

Para armazenar os dados de toda aplicação, o MySQL foi utilizado como banco de dados do *WebService*, para acessá-lo basta inserir o endereço de acesso, usuário e senha do banco, e por fim, o banco de dados do projeto que seria utilizado, tudo isso em quatro linhas

de código do Spring Boot. Durante as execuções de teste necessitava-se de uma forma de averiguar o funcionamento do banco de dados, para isso utilizou-se de MySQL Workbench[10], uma IDE para administração e desenvolvimento do banco MySQL de maneira simples e eficaz.

O *WebService* possui integrado um conjunto de páginas web que permite um controle sobre a maioria das operações do sistema, para implementá-las utilizou-se tecnologias conhecidas, como HTML (*Hypertext Markup Language*), CSS (*Cascading Style Sheets*), JavaScript, e ainda JSP (*Java Server Pages*). O HTML foi utilizado para criar os corpos das páginas, em conjunto com o CSS que ajudaria deixando a página mais estilizada, ainda quanto ao CSS, foi importado e utilizado a biblioteca de CSS do Materialize[11] que já possuía modelos prontos para elementos da página. No entanto, o HTML é simplesmente uma linguagem de marcação, e portanto, não tem a capacidade de atribuir lo ǵ ica e dinâmica para as páginas, por essa razão foi utilizado o Javascript e o JSP. O Javascript é uma linguagem de programação bastante conhecida por permitir que a página interaja com o usuário, enquanto o JSP tratou da lógica, colocando os *scriptlets* (código Java entre o HTML através de tags) é possível lidar com variáveis e operações lógicas. Para uma página web ter o suporte ao *scriptlet* ela deve ter a extensão *.jsp*.

Por fim, para se ter o uso de QR Codes para estações na aplicação web, foi usado uma API do Google Charts para gerar esse tipo de código, bastando colocar o endereço seguido pelo tipo de código a ser gerado, o tamanho, e o conteúdo. Exemplo de endereço "https://chart.googleapis.com/chart?cht=qr&chs=100x100&chl=thiago".

3.2 Metodologia do Android

Terminado o desenvolvimento do *WebService*, os outros dois ambientes podem ser implementados e integrados ao mesmo tempo. Começando pelo desenvolvimento Android, que continua com o uso da linguagem Java, porém, agora com a utilização de outra IDE, o Android Studio[12], lançado pela Google e baseado na Intellij IDEA. O Android Studio oferece ao desenvolvedor um ambiente de unificação de plataformas Android, assim como, ferramentas de análise de código, que buscam por eventuais erros, tudo para ajudar o

[10] MySQL Workbench versão 6.3.8 - https://www.mysql.com/products/workbench/
[11] Materialize 2018 - https://materializecss.com/
[12] Android Studio versão 3.1.4 - https://developer.android.com/studio/

desenvolvedor a ter maior produtividade nesta IDE. Há também um compilador flexível, o Gradle, que tem como uma de suas funcionalidades a capacidade de detectar apenas tarefas desatualizadas e executar apenas estas.

Toda aplicação Android nesta IDE apresentará três pastas importantes: (I) a pasta *manifests*, onde há o arquivo com configurações do projeto, o *AndroidManifest.xml*, nesse arquivo que serão inseridas permissões que o aplicativo venha a precisar, declaração de *activities*, *activity* principal do aplicativo; (II) a pasta *java* possui toda a lógica e controle da aplicação; (III) a pasta *res*, abreviação de *resources,* acaba sendo um conjunto de recursos para o design do aplicativo, desde cores e textos padrões até imagens e telas em XML (*Extensible Markup Language*). O funcionamento do Android Studio acaba se distanciando do utilizado pelo Eclipse, por exemplo, no Android Studio tem-se o sistema de telas conhecidas como *activities* ou *fragments* e seus métodos de funcionamento são únicos, e isso acaba sendo uma das várias particularidades desta IDE acaba deixando-a um ambiente pouco familiar nos primeiros usos.

Ainda tratando-se do Android Studio, seu modo de operar exclusivo acaba tendo seu lado bom, a praticidade com determinadas necessidades. Como é o caso do Retrofit, uma biblioteca criada pela Square, muito utilizada e que facilita as requisições HTTP, ou seja, facilita a conexão REST ao *WebService*. O interessante do Retrofit é o seu auxílio com a conversão de objetos JSON, e com os plugins certos, essa tarefa se torna automática, no presente trabalho, utilizou-se o plugin Jackson, o mesmo utilizado no *WebService*, para essas conversões. Para utilizar ambos, tanto o Retrofit quanto o Jackson, bastou-se inseri-los como dependências no Android Studio.

No projeto também foram usadas funcionalidades nativas do Android Studio, como o SQLite, um banco de dados local, que nesse projeto foi utilizado para salvar o *token* do dispositivo que é gerado na primeira execução. E também foram usadas as AsyncTasks, as classes que estendem o Async são responsáveis por executar tarefas em paralelo, como *threads*, porém com a facilidade de lidar com o pré, durante e o pós da execução. As AsyncTasks nesse projeto lidavam com o *token* do dispositivo salvo no banco local para mandar ao *WebService* e estabelecer o relacionamento entre um usuário e um dispositivo em um login, e o inverso num logout, isso ajuda a localizar os dispositivos em que um usuário está logado para enviá-lo notificações.

Para aprimorar ainda mais o Android, utilizou-se um serviço *back-end*, também desenvolvido pela Google, e que auxilia os desenvolvedores *mobile*, conhecido como Firebase[13], ele disponibiliza diversas ferramentas gratuitas, como banco de dados, análises do aplicativo, Cloud Messaging utilizado para *push notifications*, serviços de autenticação, e de anúncios, entre outros. No projeto utilizou-se apenas o serviço de autenticação e o Cloud Messaging, para usar ambos, deve-se primeiramente inserir as dependências necessárias[14], e conectar o projeto do Android Studio a algum projeto em uma conta Firebase (Google).

A autenticação foi utilizada para validar o login e então criar uma sessão para o usuário até que ele dê logout, fazendo prático o acesso ao aplicativo. Já o Cloud Messaging é utilizado para que o *WebService* consiga mandar dados para a aplicação *mobile*, mesmo quando está não fez nenhuma requisição, por essa razão pode ser usado para notificações. Para utilizar o Cloud Messaging deve-se, também, inserir um *token* de servidor (disponibilizado pelo Firebase) no *WebService*, e a partir disso, o *WebService* pode mandar mensagens para o Android.

3.3 Metodologia do Drone

Neste terceiro ambiente, por utilizar o Raspberry Pi como cerne do drone, optou-se por utilizar a linguagem padrão dele, que é o Python (versão 2.7). As principais virtudes dessa linguagem é quanto produtividade e legibilidade, ela tem uma sintaxe muito mais simples do que Java, e ainda assim, é totalmente compreensível. Python também pode oferecer diversos paradigmas de programação como orientação a objetos, procedimental, e outras, podendo ser usada desde pequenas aplicações a sistemas complexos. Por fim, assim como Java, ela possui uma comunidade muito ativa para suporte e disponibilização de diversas bibliotecas.

Para realizar a escrita do código em Python e depurações de teste buscou-se uma terceira IDE para o projeto, dessa vez foi o PyCharm[15] o utilizado. Desenvolvido pela JetBrains, com um design e funcionamento muito similar ao Android Studio, o PyCharm oferece análise de código em busca de erros de sintaxe, fácil navegação entre arquivos e janelas, entre outras ferramentas.

---

[13] Firebase 2018 - https://firebase.google.com/?hl=pt-br
[14] Dependências disponiveis em https://firebase.google.com/docs/?hl=pt-br
[15] PyCharm versão 2018.2.4 - https://www.jetbrains.com/pycharm/

Já dentro da IDE, um dos fatores primordiais para o funcionamento do drone, é a sua conexão com o *WebService* intermediário de todo sistema. Porém, apenas uma operação REST é ineficaz, pois a maioria das vezes é o *WebService* que se conectará com o drone, quebrando totalmente o princípio RESTful, onde o cliente faz requisições, e então, o *WebService* responde. E ainda, no Raspberry Pi não há o suporte de um *token* Firebase para realizar um Cloud Messaging como no Android. Por essas razões, foi pensado em utilizar *sockets*, onde tornamos o Raspberry Pi em cliente, e o *WebService* em servidor, dessa forma, uma comunicação sempre estará estabelecida enquanto um dos lados, mais precisamente o cliente, não desfizer a desconexão. A partir disso, toda vez que o *WebService* quiser enviar dados para o Raspberry Pi, ele o fará através de seu servidor, e o Raspberry Pi da mesma forma, só que através de seu cliente.

Todos esses dados recebidos, serão salvos em um banco de dados dentro do próprio Raspberry Pi, dados básicos que só dizem respeito àquele Raspberry Pi. Para gerenciar o diálogo com um banco de dados local, utilizou-se o SQLAlchemy, uma biblioteca Python muito conhecida entre os desenvolvedores, que se baseia na linguagem de banco de dados SQL, e possui a mesma função do Hibernate, mapear objetos relacionais. Todas inserções e consultas, que o Raspberry Pi realiza, passam por operações dessa biblioteca.

Já para a movimentação do drone, são necessários certos tipos de dispositivos para garantir o controle do mesmo e segurança com o meio externo. Nesse projeto foram utilizados dois tipos de sensores, e um modelo de motor. O primeiro sensor é um HC-SR04, um sensor ultrassônico, com ele pode-se emitir ondas ultrassônicas, e recebê-las, e é com o período de tempo entre emitir e receber as ondas que se calcula a distância que o drone está de um obstáculo. Já o segundo sensor trata-se de um sensor óptico reflexivo, o TCRT 5000, ele possui duas operações, emitir um feixe infravermelho, e captá-lo, dessa forma, caso ele emita sobre uma superfície branca (reflexão) ele receberá um sinal *LOW* e, em uma superfície preta (absorção) um sinal *HIGH*, esse sensor foi utilizado para fazer a leitura da linha a ser seguida, ou seja, ele que orienta o caminho do drone. Por fim, tem-se o uso dos motores DC de 3-6V, no caso, dois motores, por questão de tração para curvas e equilíbrio, para utilizá-los em conjunto a uma plataforma de controle deve-se utilizar uma ponte que conecte-os e alimente-os, no projeto usou-se o Módulo Ponte H L298N, que além de conectar o Raspberry Pi aos motores, ele também conecta um suporte para 4 pilhas AA para alimentar os motores.

Para concluir o drone, foi colocado algo para garantir o funcionamento do sistema de estações por leitura de QR code, a começar pelo uso de uma câmera, para o projeto utilizou-se um módulo de câmera para Raspberry Pi de 8MP. Em conjunto com ela, foram utilizadas duas bibliotecas para fazer o processamento de imagem, a primeira biblioteca é famosa pelo seu emprego para visão computacional em Python, o OpenCV, em paralelo empregou-se outra biblioteca, dependente de OpenCV para seu funcionamento, a ZBAR, com a função de detectar e ler os mais diversos códigos de barras existentes.

4 Desenvolvimento

Como mencionado na metodologia, o desenvolvimento do sistema começou a partir da elaboração de esquemas de funcionamento e casos de uso, esses que são essenciais para que o projeto não seja custoso quanto a reparos conforme for avançando. Dentro dos casos de uso, busca-se explorar o máximo de funções que o sistema pode desempenhar.

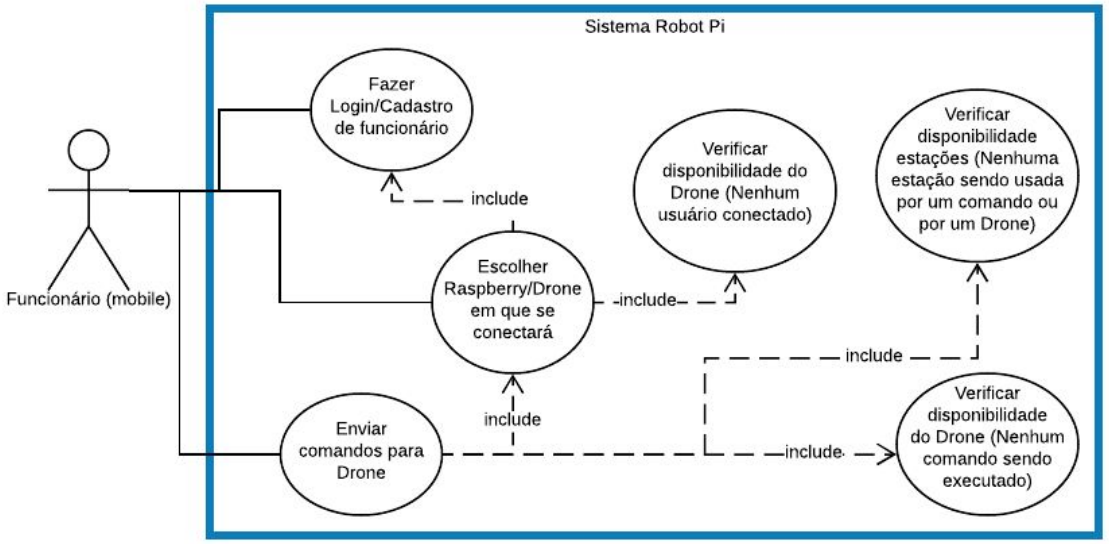


Figura 3 – Diagrama de casos de uso para o funcionário

Fonte: Autoria própria

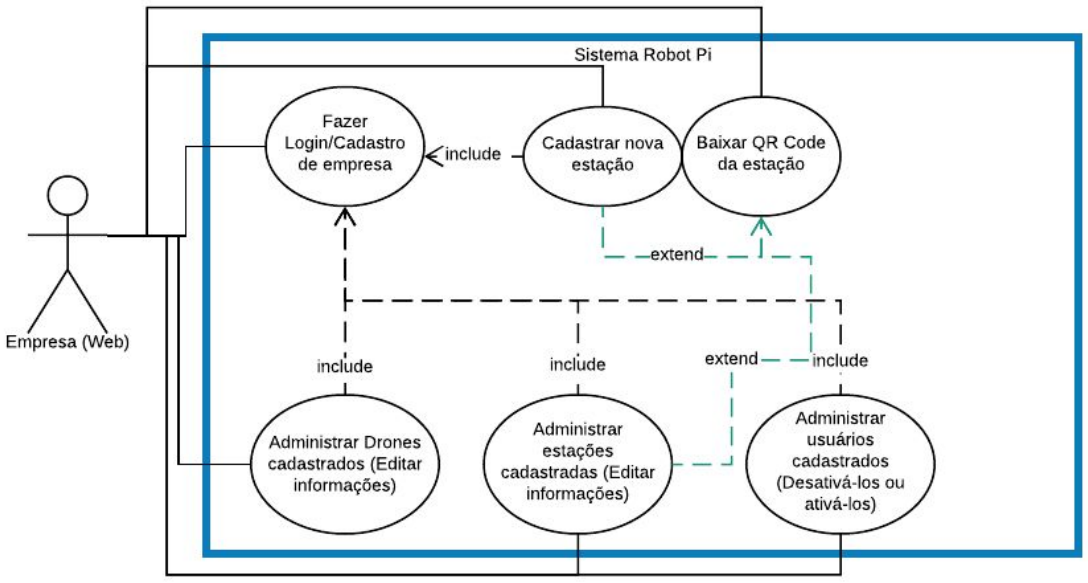


Figura 4 – Diagrama de casos de uso para a empresa

Fonte: Autoria própria

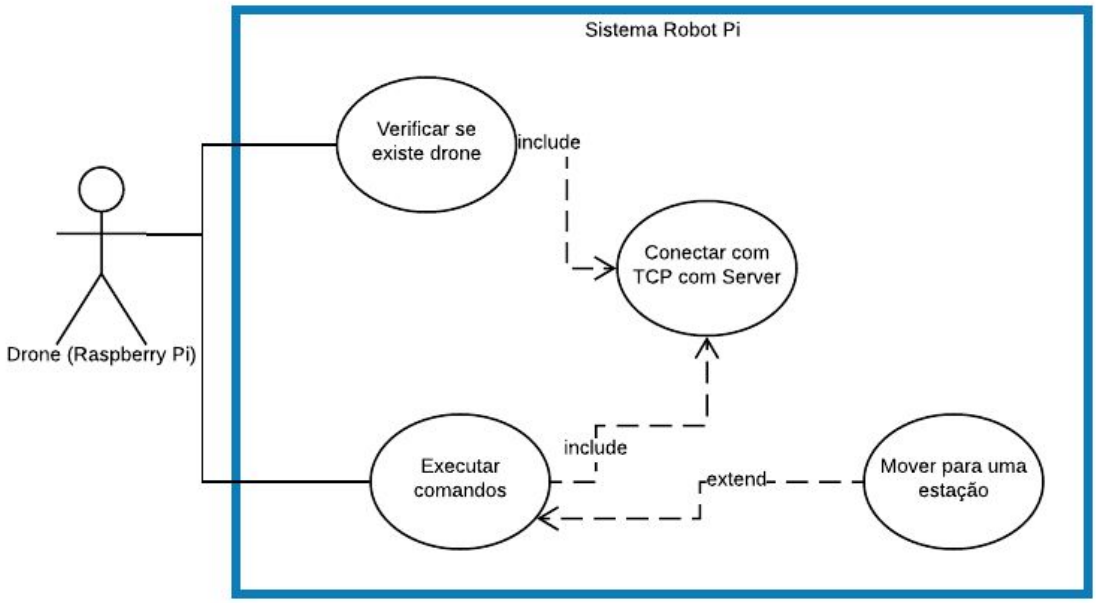


Figura 5 – Diagrama de casos de uso para o drone

Fonte: Autoria própria

Como demonstrado nas três figuras, temos de forma resumida o funcionamento do projeto. Optou-se por dividir em três quadros de casos de uso pensando no entendimento, já que são três ambientes distintos que se integram.

Como pode ser visto pela Figura 3, o processo de trabalho do aplicativo *mobile* para o funcionário é relativamente simples, pois ele simplesmente escolherá um drone para se

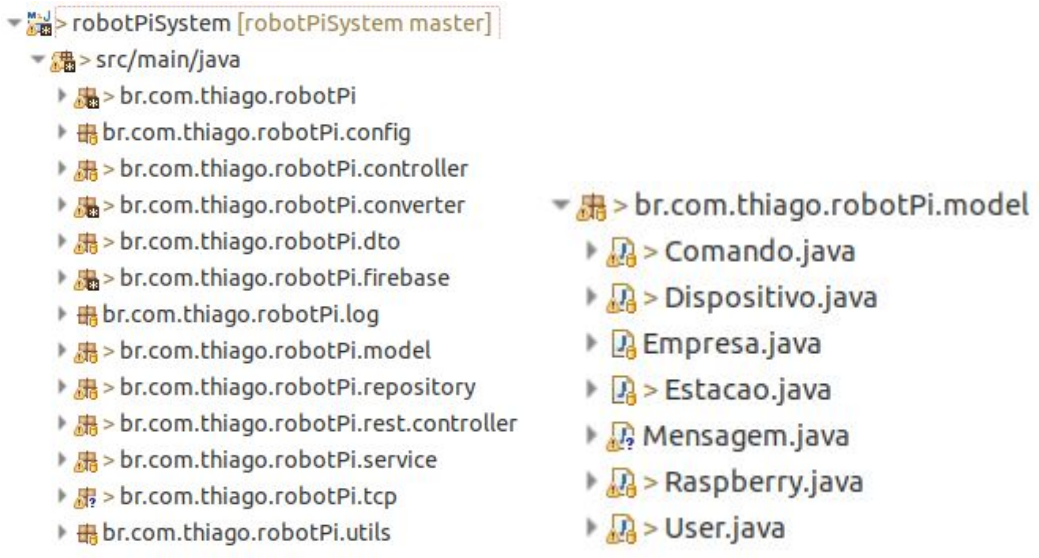
conectar, e caso sua conexão ocorra, terá liberdade para montar e enviar comandos para o *WebService* que gerenciará tudo.

Quanto ao *WebService* da Figura 4, ele se refere mais especificamente ao funcionamento referente à empresa pelo sistema web, ou seja, todas as operações que uma empresa tem alcance de realizar. Nota-se a capacidade de gerenciar todos os usuários (funcionários) pertencentes a essa empresa para permitir-lhes ou não o acesso, assim como as informações dos drones, além de poder gerar novas estações. No entanto, esse sistema web não foi implementado nessa primeira versão do projeto, então suas operações são realizadas, por enquanto, através de uma aplicação web para controle de todo sistema.

Já o drone, representado na Figura 5, toda vez que for iniciado verificará se já possui um drone salvo no seu banco de dados local, caso não haja, ele enviará ao servidor um comando para criar um novo e ser retornado, se houver um drone salvo, então será feito uma requisição por informações novas que possam haver referentes ao drone e atualizá-lo com elas. Para os comandos ocorre o contrário, o servidor que envia para o Raspberry Pi o comando para executar.

4.1 Desenvolvimento do *WebService*

Após finalizada a parte de estruturação do projeto, começou-se a implementá-lo, para isso foi escolhido não começar pelo início, nem pelo fim, mas pelo meio. O *WebService* é o intermediário de toda aplicação, ele que cuida de todo armazenamento de dados e lógica da aplicação. Durante seu desenvolvimento ele acabou abrigando muito mais do que operações RESTful, ele passou a rodar uma aplicação web simples para o controle do projeto, e um servidor com protocolo TCP/IP para conectar-se via *socket* aos drones. E como mencionado anteriormente, nessa versão não foi implementado a aplicação web para as empresas operarem, como havia sido pensado, pois não haveria tempo suficiente para os devidos testes das funcionalidades básicas.



(a) (b)

Figura 6 – (a) Organização Webservice e (b) pacote Model

Fonte: Autoria própria

Para uma melhor organização do projeto, optou-se por dividir as funcionalidades do *WebService* em várias pacotes e arquivos, Figura 6.

A primeira parte a ser implementada no *WebService*, por se tratar de um projeto Spring Boot, é o arquivo que contém a *main* do projeto, o *Application.java*. Ele está localizado no pacote raiz (robotPi) e é ele quem deverá executar todo projeto, para isso, ele recebe uma série de anotações do Spring, como o *@SpringBootApplication* e *@EnableAutoConfiguration*, ambas são muito importantes para algumas configurações básicas do Spring, e temos também a *@EnableAsync*, que permitirá a implementação de tarefas assíncronas. Nesse mesmo arquivo ainda há um método com algumas configurações necessárias das tarefas assíncronas, ele é anotado como *@Bean* e isso o torna uma definição do projeto que será resgatado quando a aplicação for executada.

Com o arquivo principal pronto, a aplicação já pode operar, porém, sem função alguma, para que possa começar a funcionar deve-se inserir algumas tabelas no banco de dados e acessá-las. Os arquivos referentes às tabelas do banco de dados estão no pacote *model*, ele que representa todas as classes de entidades necessárias para o funcionamento do sistema. Todas essas classes são anotadas como *@Entity*, com isso, o Hibernate criará a classe como tabela no banco de dados quando a aplicação for executada, e seus respectivos atributos, caso possuam *getters* e *setters*, viram colunas destas tabelas. Além disso, essas classes

precisam de uma segunda anotação, a *@Id*, porém, esta irá sob um de seus atributos, o qual representará a chave primária da tabela.

```
@Entity
public class User {

    @Id
    @GenericGenerator(name = "id", strategy = "uuid2")
    private String id;
    private String nome;
    private String senha;
    @ManyToOne
    private Empresa empresa;
    private String telefone;
    private String email;
    private int desativado;
```

Figura 7 – Exemplo de classe anotada com *@Entity*

Fonte: Autoria própria

Como mostrado na Figura 7, junto com o *@Id* é colocado o *@GenericGenerator*, que representa um método customizado para gerar as chaves primárias das tabelas, neste projeto utilizou-se o UUID (*Universally Unique IDentifier*) versão 4, o Identificador Único Universal, ele é gerado como uma *String* de 32 caracteres hexadecimais aleatórios, divididos por hífens em cinco grupos (ex: 3d0ca315-aff9–4fc2-be61–3b76b9a2d798). Esse tipo de chave primária é muito útil em sistemas distribuidos, em que não precise de uma central para gerá-los, e sua quantidade de variações acaba por quase impossibilitar que uma chave primária seja duplicada no sistema.

Outra anotação muito importante presente na Figura 7 é quanto as cardinalidades, caso alguma classe possua outra, como é o caso do “User” ter um atributo do tipo “Empresa”, isso deve ser anotado para que o Hibernate consiga fazer a inserção de chaves estrangeiras. No caso citado, um usuário (Funcionário) pertence exclusivamente a uma empresa, e uma empresa poderá ter muitos usuários, dessa forma, tem-se uma cardinalidade de muitos para um, e por isso, anota-se o atributo referente a outra classe como *@ManyToOne*, também há casos de atributos *@OneToOne*, onde não depende de qual tabela ficará a chave estrangeira.

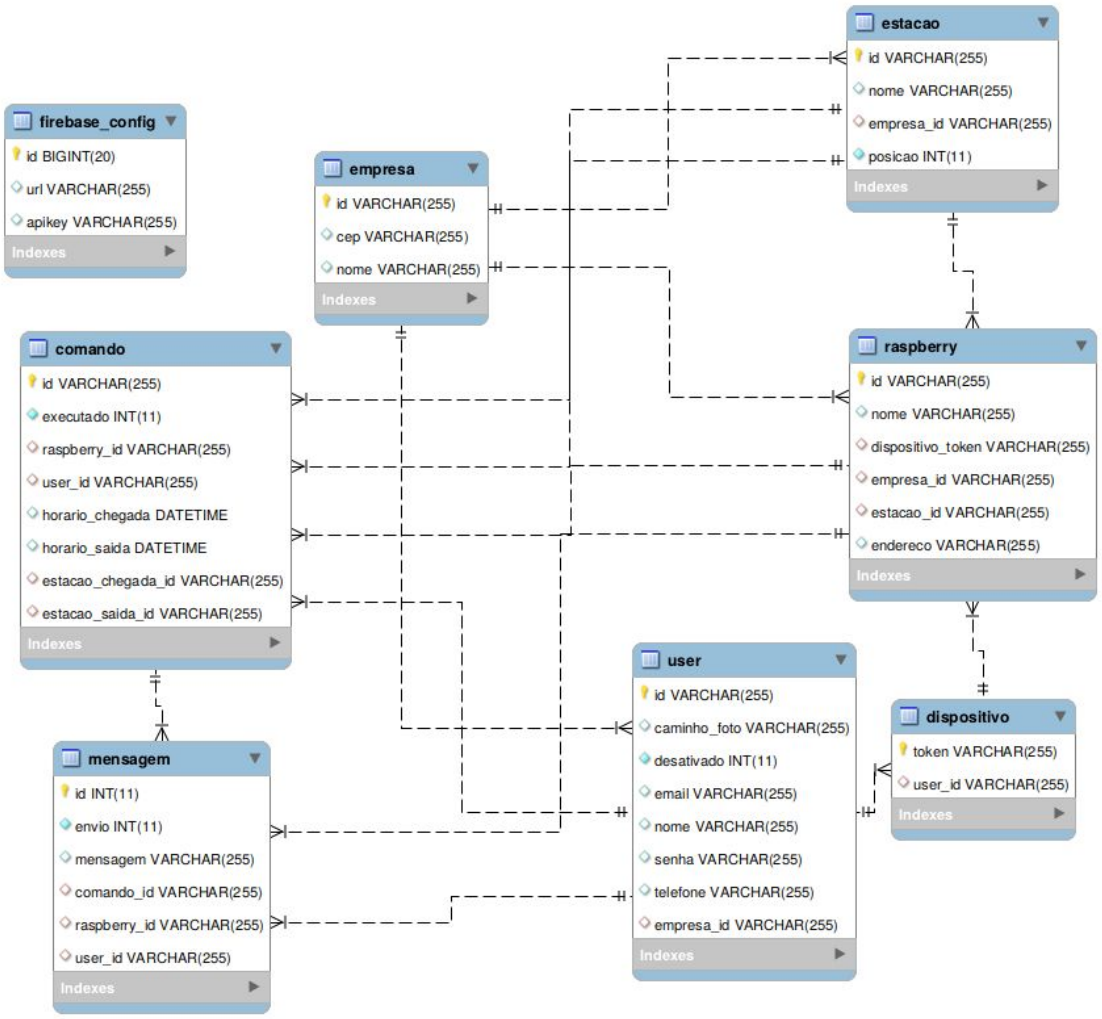


Figura 8 – Diagrama Entidade-Relacionamento do projeto

Fonte: Autoria própria

Na Figura 8 pode-se notar um diagrama ER de todas as tabela da aplicação, estas geradas a partir das classes anotadas pelo Spring junto as suas colunas, e suas chaves primárias e estrangeiras.

4.1.1 Bases do *WebService*

Com tudo isso feito, tem-se um banco de dados montado, mas ainda não há um *WebService* funcional e que seja capaz de realizar consultas nesse banco de dados. Por essa razão, devem ser implementadas as funcionalidades de três pacotes diferentes, o *repository*, o *service*, e o *rest controller*.

As interfaces do pacote *repository* são relacionadas às consultas que serão feitas ao banco, cada uma das entidades possui uma interface *repository* própria, como por exemplo, “UserRepository”, “EmpresaRepository”, “ComandoRepository”, e assim por diante. Estas interfaces precisam apenas estender a classe “PagingAndSortingRepository”, indicar uma entidade e o tipo da chave primária da mesma. Diferente da maioria das classes, estas interfaces não precisam de anotações para seu funcionamento, a não ser em cada um dos seus métodos de consulta. Nesse caso, uma anotação *@Query* deve ser feita, e ainda receber como parâmetro uma *String* contendo a *query* de consulta. Ademais, pode-se ter uma anotação *@Param*, se um parâmetro do método for utilizado como uma variável de condição na *query*, dessa forma, deve-se indicar qual será o nome que será usado na *query* para acessá-lo.

```
public interface UserRepository extends PagingAndSortingRepository<User, String> {

    @Query("FROM User u WHERE u.desativado = 0")
    List<User> visiveis();

    @Query("FROM User u WHERE u.email = :email AND u.desativado=0")
    User findUser(@Param("email") String email);

    @Query("FROM User u WHERE u.empresa = :empresa")
    List<User> usersFromEmpresa(@Param("empresa") Empresa empresa);
}
```

Figura 9 – Exemplo de interface com suas consultas

Fonte: Autoria própria

O funcionamento das classes do *repository* pode ser exemplificado na Figura 9. Os métodos que são implementados nessas interfaces são consultas ao banco que são necessárias para o funcionamento do projeto, porém, o Hibernate já oferece alguns métodos próprios só por estender a classe “PagingAndSortingRepository”, como métodos de inserção, atualização, busca por chave primária, entre outros.

Já no pacote *service* são implementadas classes que possuem funções de lógica na aplicação, fazendo comparações, laços, e realizando as buscas pelas interfaces do pacote *repository*. Todas as classes *services* são referentes a uma entidade também, como por exemplo, “ComandoService”, “RaspberryService”, “UserService”, e assim por diante. Nesse pacote, as classes que operam como *services* devem ser anotadas como *@Service*, pois, é em uma classe *service* que se terá acesso a outros serviços da aplicação, por exemplo, se um

serviço de “Comando” quer verificar se um “Raspberry” continua conectado ao sistema, ele chama um método do “RaspberryService” para fazer essa consulta.

```java
@Service
public class UserService {

    private UserRepository userRepository;
    private DispositivoService dispositivoService;
    private static final Logger LOGGER = Logger.getLogger(UserService.class);
    private UUIDUtils uuidUtils;

    @Autowired
    public UserService(UserRepository userRepository, DispositivoService dispositivoService,
            UUIDUtils uuidUtils) {
        this.userRepository = userRepository;
        this.dispositivoService = dispositivoService;
        this.uuidUtils = uuidUtils;
    }
```

Figura 10 – Exemplo de classe do pacote *service*

Fonte: Autoria própria

Na Figura 10 observa-se ainda a presença da anotação *@Autowired* no construtor, isso permite que esse construtor seja preenchido de forma “automática” na execução da aplicação, sem que as classes que forem instanciar um *service* precisem passar um enorme quantia de parâmetros.

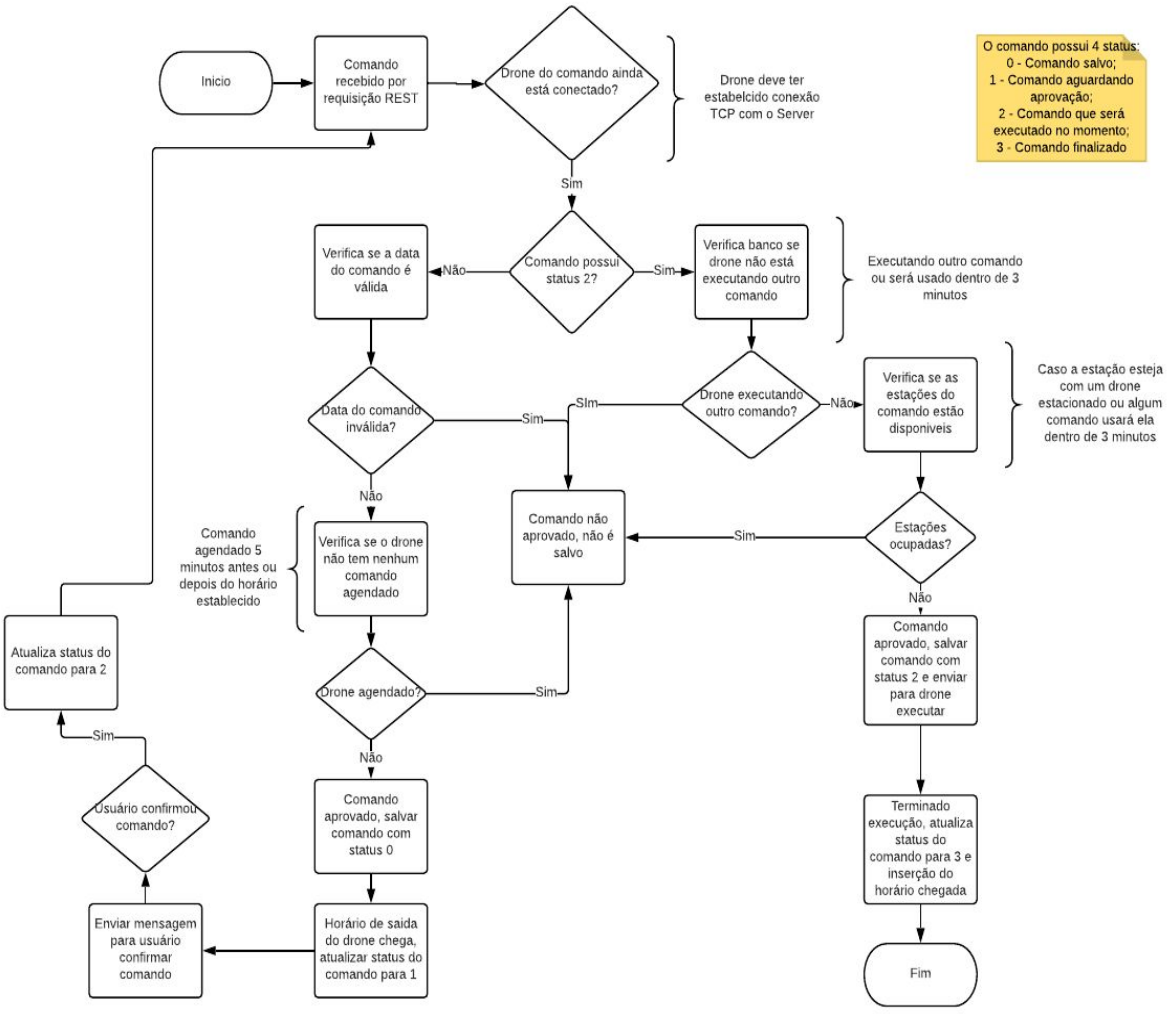


Figura 11 – Diagrama do fluxo de comandos

Fonte: Autoria própria

Diferente da maioria dos *services* que possuem métodos padrões, como de consultas e inserções de objetos, o "ComandoService" acaba sendo um exclusivo, tendo uma importante tarefa no sistema, coordenar todos os comandos que chegam, que saem, que são executados, que estejam executando, entre outros, como pode ser visto na Figura 11. Um de seus métodos é executado como assíncrono assim que a aplicação inicia, sua função é ficar dentro de um *loop* infinito buscando por comandos que estão agendados para o horário atual, caso ache algum, usa-se o FirebaseSender (abordado na Seção 4.1.3) para enviar uma mensagem de confirmação ao usuário nos dispositivos em que está logado. Enviar mensagens ao usuário também é uma operação assíncrona em qualquer ocasião em que é utilizada, para que não se

pare todo processo apenas para uma tarefa simples. Por se tratar de Spring, para um método ser assíncrono basta anotá-lo como *@Async*.

Agora no último pacote, para que o sistema RESTful funcione perfeitamente, precisa-se de um local que receba e saiba direcionar as requisições REST, essa é a função do pacote *rest controller*. Nele há classes anotadas como *@RestController* preparadas para receber requisições, não bastando apenas isso, cada classe ainda tem uma anotação *@RequestMapping*, passando uma *String* com um endereço para essa classe esperar a requisição. As classes comportam diversos métodos, e cada método recebe igualmente uma anotação *@RequestMapping*, pois ele também só é utilizado quando determinado endereço é acessado, contudo, ele necessita de alguns outros parâmetros, como o método REST utilizado (POST, GET, PUT, etc.), o que ele vai receber (JSON), o que ele vai produzir (JSON), esses dois últimos são opcionais e dependem do método REST escolhido.

```
@RestController
@RequestMapping("api/user")
public class UserRestController {

    private UserService userService;

    @Autowired
    public UserRestController(UserService userService) {
        this.userService = userService;
    }

    @RequestMapping(method = GET)
    public @ResponseBody UserSync lista() {
        return userService.getSyncLista();
    }

    @RequestMapping(method = POST, consumes = JSON, produces = JSON)
    public @ResponseBody UserSync insereOuAltera(@RequestBody User user,
```

Figura 12 – Exemplo de classe do pacote *rest controller*

Fonte: Autoria própria

A Figura 12 mostra alguns detalhes, como o uso de um *service* sem precisar instanciá-lo manualmente graças ao *@Autowired*, e o uso do mesmo no construtor do *rest controller*. Ainda observa-se o uso de mais duas anotações, a *@ResponseBody* que devolverá um "Response" para o Android tratar, e o *@RequestBody* que obriga o envio, por parte do cliente, de um objeto no *body* da requisição.

4.1.2 Funcionalidades Complementares

Com os pacotes básicos já operantes, tem-se um *WebService* funcional, mas ainda não no seu melhor desempenho e organização. Para isso foram implementados os pacotes *dto*, *converter* e *firebase*.

O pacote *converter* acaba sendo o mais simples, pois terá apenas um arquivo com uma classe de conversão anotada como *@Converter* para manipular o objeto "Date" que a entidade "Comando" tem, pois, para inserir no banco deve ser convertido para "Timestamp", e ao resgatar do banco, convertido novamente em "Date". Esta classe deve implementar "AttributeConverter" e deve estipular os dois objetos que serão manipulados ("Date" e "Timestamp").

Seguindo para o pacote *dto*, que representa a sigla DTO (*Data Transfer Object*), ele diz respeito aos objetos que são retornados nas requisições REST. As classes desse pacote possuem um objeto e uma lista de objetos da mesma entidade, como o "UserSync" no método "lista()" na Figura 12, ele possui um atributo do tipo "User" e outro atributo do tipo "List<User>". Foi escolhido esse tipo de objeto, pois pode-se aproveitá-lo para diversas situações, como o envio de apenas um ou o envio de uma lista, e caberá ao Android verificar como que esse DTO está preenchido para utilizá-lo. Assim como nos outros pacotes apresentados, as classes são relacionadas as entidades, e possuem os nomes seguidos por *Sync*, "RaspberrySync", "EmpresaSync", "EstacaoSync", e assim sucessivamente.

Para finalizar esta parte de aperfeiçoamento do projeto, ainda há o pacote *firebase*, esse sendo um dos mais vitais por tratar do envio de mensagens sem requisição por parte do usuário. Dentro dele há cinco arquivos, o primeiro arquivo é o *FirebaseConfig.java* que possui uma simples classe anotada com *@Entity*, pois é nela em que fica salvo o *token* de servidor que recebe-se do Firebase para o uso do Cloud Messaging, como visto na Seção 3.3. Já o segundo arquivo é o *CloudMessaging.java*, nele há uma classe de mesmo nome responsável por associar um JSON de um objeto DTO a um dispositivo destino para o qual será enviado. O arquivo *FirebaseSender.java* é a central de envio, é ele quem tem a função de preencher o objeto "CloudMessaging" e enviá-lo para os dispositivos em que o usuário esteja logado, como já visto, porém, para enviá-lo deve-se montar uma requisição, e essa é a função do quarto arquivo. O *FirebaseClient.java*, que recebe uma *String* do objeto

"CloudMessaging", e monta um objeto de requisição ("Request"), colocando junto o *token* que autoriza o envio. Por fim, tem-se um quinto arquivo, o *PrepareToSend.java*, com a simples função de retornar um objeto do "FirebaseSender" para qualquer *service* que precise enviar algo para um usuário.

As classes do *firebase* operam no sistema para as funções de envio de mensagens aos usuários, mas também são utilizadas para o envio de uma lista de drones com o status atualizado para os usuários da mesma empresa, isso ocorre toda vez que um Raspberry Pi se conecta ao sistema ou um usuário se conecta a um Raspberry Pi.

Por fim, foi criado um pacote para gerenciar a parte do servidor com protocolo TCP/IP, nele há apenas uma classe, que é executada em paralelo a aplicação principal, e é ela a responsável por criar o servidor, escutar as requisições dos clientes (Raspberry Pi), escrever para os clientes, e gerenciar todas as conexões e desconexões.

4.1.3 Controle do Sistema

Para se ter um controle sobre os processos que ocorrem no sistema de um modo intuitivo pensou-se na criação de um conjunto de páginas web que rodassem no próprio *WebService*. Para implementá-las foram utilizadas tecnologias comuns para o desenvolvimento web, como HTML, CSS e Javascript, e para torná-las dinâmicas utilizou-se o JSP.

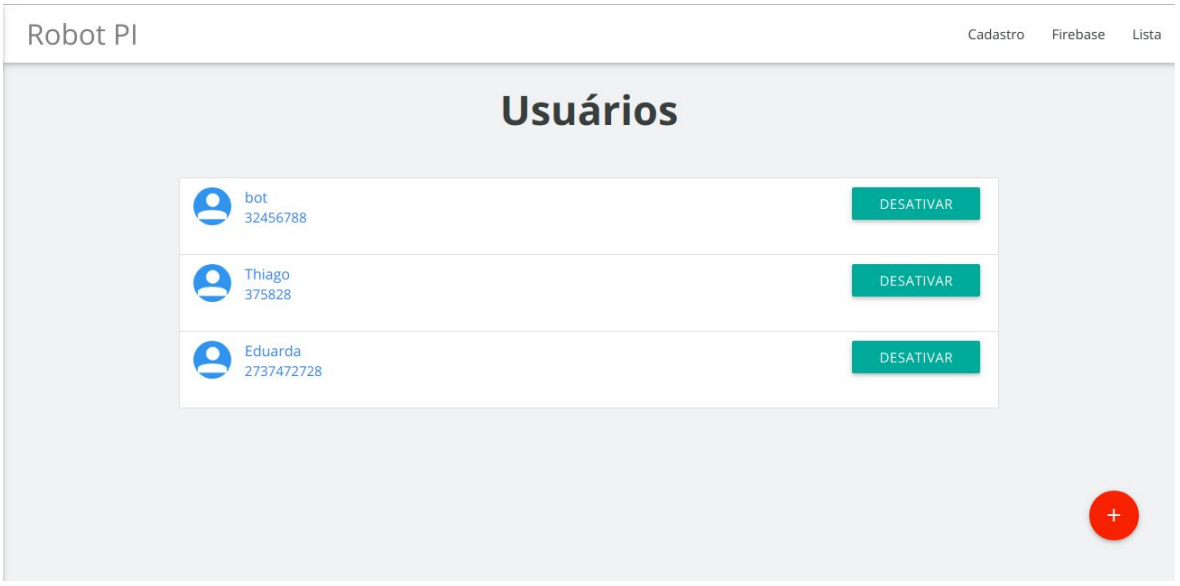


Figura 13 – Página com lista de usuários de uma empresa

Fonte: Autoria própria

Porém, o JSP sozinho não pode fazer as consultas e direcionamentos de páginas, para isso foi implementado o pacote *controller*, que é responsável pela consulta e envio de objetos para preencher as páginas durante a renderização, e também responsável pelo retorno das páginas já renderizadas para o navegador. Um exemplo disso tudo é a página de listagem de usuários de uma empresa, ao clicar numa empresa na lista de empresas, uma função do "UserController" é chamada, nela é feita a consulta pela lista de usuários daquela empresa, depois, essa lista é adicionada a página web que deve conter a lista de usuários, esses que também podem ser acessados e editados, Figura 13.

4.1.4 Criação do Projeto Firebase

Para algumas funcionalidades importantes tanto do Android quanto do *WebService* é necessário o uso do Firebase. Para utilizá-lo é simples, basta conectar uma conta Google, e criar um projeto, em seguida tem-se acesso a diversas funcionalidades já mencionadas. Nessa etapa aproveita-se para ir até as configurações do projeto no Firebase e copiar o *token* do servidor para o uso no *WebService*.

4.2 Desenvolvimento do Android

Com o *WebService* funcionando, a parte Android pôde ser implementada e testada. No princípio, foram montados todos os *layouts* de telas e botões que seriam necessários, todos inseridos na pasta *res* do projeto no Android Studio. Foram definidos também os ids para os elementos das telas, como os *inputs* e botões, dessa forma, pode-se acessá-los e verificar o texto escrito ou quando há um clique de botão. Porém, apenas com os componentes do *res* não se tem o suficiente para testar a aplicação, a invocação de telas, por exemplo, fica na pasta *java*, assim como toda a lógica da aplicação.

O primeiro passo no Android Studio é integrar nossa aplicação ao projeto no Firebase, em seguida, escolhe-se quais serviços Firebase serão utilizados (Autenticação e Cloud Messaging) e segue-se as respectivas instruções.

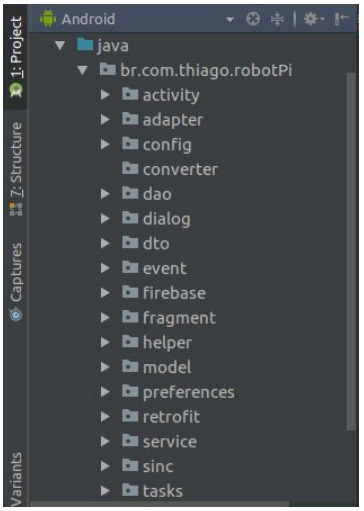


Figura 14 – Organização do projeto no Android Studio

Fonte: Autoria própria

Assim como no Eclipse, foi utilizada a mesma lógica de organização para o Android Studio, diferentes pacotes para diferentes funcionalidades, mais especificamente na pasta *java*, Figura 14.

O pacote *activity* é o principal em qualquer aplicação Android, é ele quem movimenta toda a aplicação, e dentro do mesmo encontram-se os arquivos de lógica de cada tela montada previamente. Eles têm acesso ao momento de criação da tela, de destruição, de pause, de retorno, entre outros, por essa razão, cada tela deve estar relacionada a uma *activity* obrigatoriamente.

A primeira *activity* é a tela de login, ela foi definida no *AndroidManifest.xml* como tela de lançamento, ou seja, a primeira *activity* a ser executada pela aplicação. Esta *activity* tem como função verificar, no momento em que é iniciada, se já existe uma autenticação feita, para isso existe o pacote *config*, ele conta com classes Firebase específicas que retornam dados de um usuário logado se houver. Caso haja uma autenticação, a *activity* de carregamento já é invocada automaticamente, senão, a tela de login fica aberta esperando que o usuário logue-se ou registre-se. O usuário, para logar, preenche seu e-mail e senha, e então é verificado se esses dois dados estão nas autenticações do projeto no Firebase, se não existirem lá, é porque a conta não existe na aplicação, caso contrário, é criada uma autenticação naquele dispositivo que permite a entrada sem logar novamente.

No caso do usuário não ter uma conta ainda, ou tentar logar e ocorrer erro de conta não existente, invoca-se a "CadastroActivity", esta tela possui campos como nome, senha, e-mail, telefone, CEP, e empresa, todos são obrigatórios para o preenchimento do objeto "User". Nesta *activity* há dois usos de requisições REST para o *WebService*, o primeiro é um GET de todas as empresas logo na criação da tela, e o segundo é um POST do objeto "User" montado para ser salvo. Ao criar a conta com sucesso, é feita a inserção desse usuário nas autenticações do projeto no Firebase, e também é realizado o POST de inserção no banco de dados da aplicação, com um detalhe importante, o usuário está com o atributo desativado ativo, ou seja, a empresa deverá ativá-lo para que ele possa logar futuramente com sucesso na aplicação.

Para realizar as requisições mencionadas têm-se dois pacotes importantes, o *retrofit*, que possui uma classe "RetrofitInicializador", responsável por configurar toda a requisição, desde endereço de conexão até o uso do Jackson, e o outro pacote é o *service*, que contém as interfaces com as consultas REST feitas em toda aplicação, estas interfaces são retornadas por métodos dentro da classe do "RetrofitInicializador". Outro ponto a ser notado é a presença do pacote *dto*, quando uma requisição retorna algo do *WebService* é um DTO, e como já visto na Seção 4.1.2, caberá ao Android lidar com esse objeto da maneira necessária.

Com êxito no login, chega-se a tela de carregamento ("LoadActivity"), ela fará algumas preparações antes de invocar a *Main*. Uma dessas preparações é carregar todo o objeto "User" do banco, pois, até o momento, só se verificou sua existência pelo Firebase. Para carregá-lo é feita uma requisição GET passando como parâmetro o e-mail do usuário autenticado no Firebase, e em caso de sucesso, o "User" é passado para a "MainActivity" através do próprio objeto de invocação de telas ("Intent"). Se algo der errado no processo, é mostrado um erro que redireciona para a tela de login.

A "MainActivity", por sua vez, representa a página principal da aplicação, quando criada, ela executa uma "AsyncTask" (pacote *tasks*) de envio do "User" juntamente com o *token* do dispositivo em que está logado, representando o funcionamento explicado na Seção 3.2. Em seguida são estabelecidas duas abas, uma aba para a listagem dos drones, e a outra para o histórico de comandos. Para o funcionamento destas abas foram utilizados *fragments*, com isso, pode-se ter diversos comportamentos em uma mesma *activity*.

(a) (b) (c)

Figura 15 – (a) Lista de drones, (b) tela de comandos e (c) exemplo de montagem de comando

Fonte: Autoria própria

O "RaspberryFragment" é o responsável pela aba de listagem dos drones, ele está no pacote *fragment* assim como os outros *fragments* da aplicação. Ele é criado logo que a "MainActivity" estabelece suas abas, e sua primeira tarefa é preparar a lista, para isso, utilizou-se o RecyclerView e o pacote *adapter*. O RecyclerView é um *layout* de lista mais "sustentável" para a aplicação, e para montar e gerenciar cada item dessa lista é necessário um *adapter*, nesse caso, o "RaspberryAdapter", Figura 15. Nesse *fragment* também é feita uma requisição GET passando a empresa do usuário, dessa forma, um DTO contendo a lista de drones da mesma empresa é enviado de volta, e então passado para o "RaspberryAdapter" colocá-los no RecyclerView. Ainda no "RaspberryFragment" há mais duas funções, atualizar os status dos drones e conectar o usuário a um.

Para a atualização da lista de drones sempre que houver uma mudança de status em um dos drones, começa-se por implementar uma função no "RaspberryFragment" responsável por atualizar a lista quando for chamada pelo EventBus (Biblioteca para atualizar sem *swipe* na tela). Também é utilizado o pacote *firebase*, ele possui uma classe com métodos específicos para receber uma mensagem JSON de um DTO do *WebService* e convertê-lo para um objeto. Nesse último método, também é criado um objeto do tipo "EventBus", e chamado seu método post, que recebe como parâmetro o objeto "RaspberrySync" (DTO). Esse

“RaspberrySync” não é passado dessa forma “crua”, pois o método no “RaspberryFragment” espera um parâmetro do tipo “AtualizaRaspberriesEvent” (pacote *event*), esse que possui um atributo “RaspberrySync”, que deve ser preenchido, para então ser enviado ao *fragment* para atualizar a lista.

Já para a conexão, só ocorre quando há um clique em um dos drones da lista, nessa situação, é chamada uma função do “RaspberryAdapter”, responsável por retornar o objeto “Raspberry” clicado, caso esse Raspberry Pi esteja conectado e sem nenhum usuário utilizando, a conexão é feita, para isso é enviado por POST o objeto “Raspberry” já com o “User” que vai se conectar. Com sucesso na conexão, o drone passa a ter o status ocupado e atualiza-se em todos os dispositivos com usuários da mesma empresa logados, por essa razão, que os usuários dessa aplicação sempre devem estar conectados à internet, preferencialmente via uma rede wi-fi.

Por fim, chega-se a última *activity* do processo, a “ComandoActivity”, ela apresenta diversas funcionalidades, a começar com uma requisição GET de mensagens entre o usuário e o drone, e para inseri-las foi utilizado o mesmo mecanismo da lista de drones, um RecyclerView e o “MensagemAdapter” para a montagem de cada mensagem, ele que organiza as mensagens em dois tipos, as enviadas pelo usuário e as enviadas pelo *WebService*. Outra funcionalidade desta *activity* é justamente enviar um comando, para isso, o usuário abre uma “tela” de montagem do “Comando”, esta tela é um *fragment* personalizado capaz de ser aberto sob a mesma *activity*, em seguida, é solicitado que escolha a estação de saída, a estação de chegada (ambas solicitações de estações utilizaram um RecyclerView e o “EstacaoAdapter”), e o horário de saída, ao clicar no botão de pronto, um objeto “Mensagem” é montado e passa a conter o objeto “Comando”. Um objeto “Mensagem” pode conter um objeto “Comando”, pois, caso um “Comando” venha a solicitar permissão para execução, a “Mensagem” saberá qual “Comando” foi autorizado. O “Comando” é enviado e passa pelo processo da Figura 11 descrita na Seção 4.1.

O sistema para receber as mensagens obedece a lógica da atualização da lista de drones, já que no *WebService*, a montagem da “Mensagem” de resposta se dá um processo assíncrono e não retorna na mesma requisição REST. As únicas mudanças são quanto ao objeto DTO, que é um “MensagemSync”, e o objeto de evento, que é o “AtualizaChatEvent”.

Vale lembrar que todas as entidades do banco de dados possuem classes no mesmo formato criadas na aplicação Android, e estão presentes no pacote *model*.

4.3 Desenvolvimento do Drone

Com o ambiente do funcionário e o ambiente de controle geral prontos, a última das três partes começou a ser implementada, o drone. No início foram feitas diversas pesquisas sobre a montagem de um veículo contendo Raspberry Pi e utilizando todos os componentes listados na Seção 3.3. Após adquirir uma base sobre o drone, começou-se a montagem do mesmo, dividindo-a em montagem física e montagem do software, que apesar de serem duas partes, sempre foram desenvolvidas simultaneamente.

4.3.1 Montagem Física

O primeiro ponto da montagem foi conseguir um chassi para que se pudesse posicionar todas as peças em cima, incluindo as rodas. Esse chassi foi adquirido pronto, feito em acrílico e com diversos furos para quaisquer tipos de montagem. Nesse mesmo chassi havia montado antecipadamente os motores DC com suas rodas, bastava então conectá-los ao Raspberry Pi.

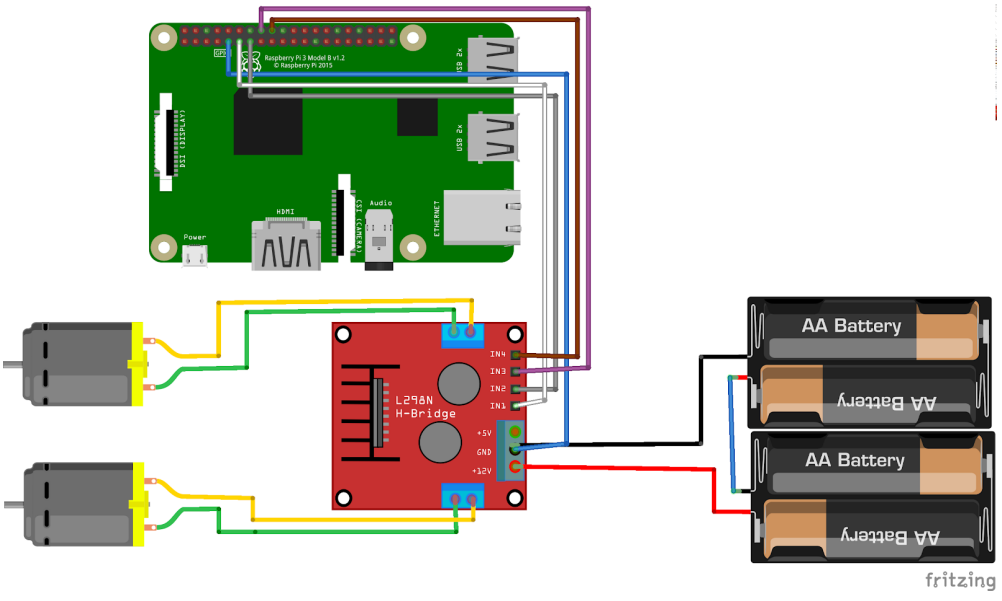


Figura 16 – Esquema de montagem dos motores

Fonte: Autoria própria

O intermediário entre os motores e o Raspberry Pi é o Módulo Ponte H L298N, nele há quatro pinos de entrada, cada um deve ser conectado a um pino GPIO do Raspberry Pi, neste projeto foram escolhidos os pinos GPIO 17, GPIO 23, GPIO 24, GPIO 27, de acordo com a Figura 2. Cada pino de entrada no Módulo Ponte possui uma porta de saída, e em cada porta de saída é conectado um dos fios dos motores, esses fios representam as polaridades dos motores, no total tem-se quatro polaridades, motor direito para frente, motor direito para trás, motor esquerdo para frente e motor esquerdo para trás, cada pino GPIO tem controle sobre uma dessas polaridades. Por fim, o Módulo Ponte recebe dois fios, um para alimentação e outro para o terra (12V e GND), vindos do suporte de pilhas AA, e também, um fio terra vindo do Raspberry Pi. A Figura 16 representa a montagem dos motores.

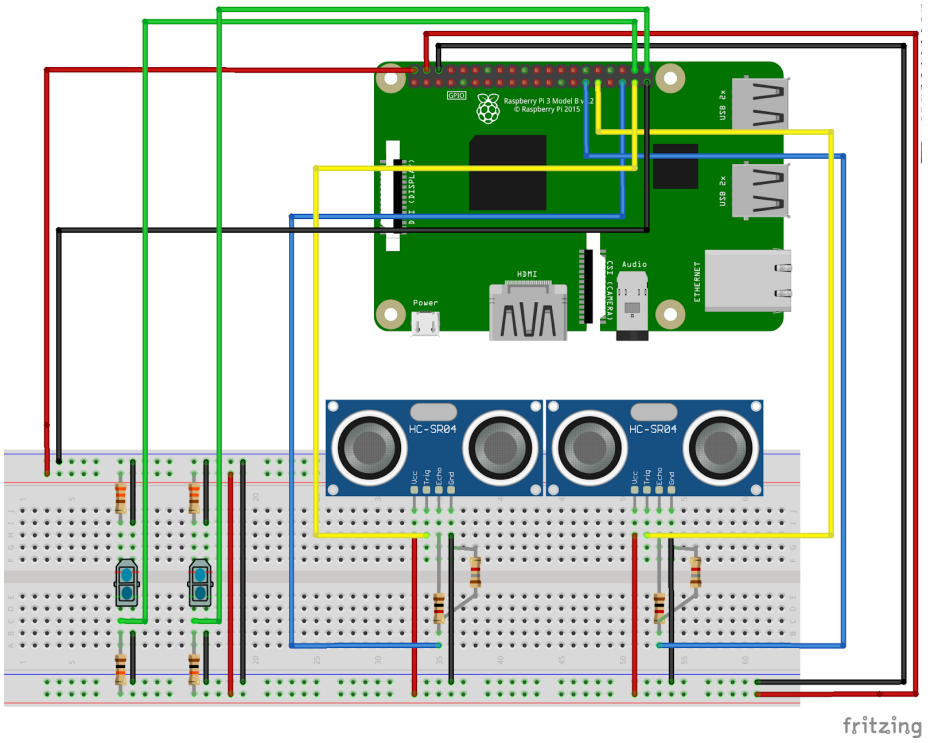


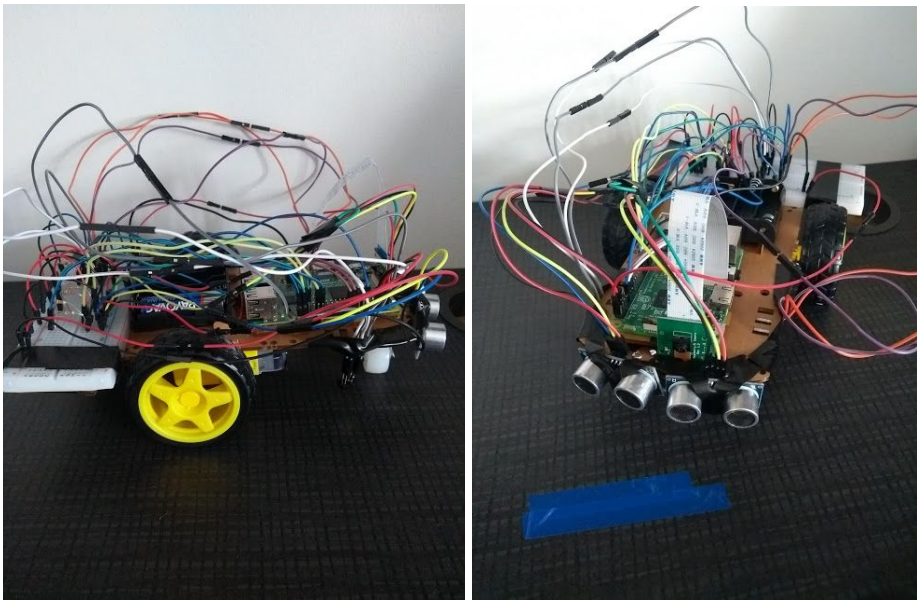
Figura 17 – Esquema de montagem dos sensores

Fonte: Autoria própria

A próxima etapa da montagem do drone foi a instalação dos dois sensores ultrassônicos HC-SR04, nesse tipo de sensor temos quatro pinos, um alimentação (VCC), um terra (GND), um *trigger*, e um *echo*, os dois últimos para emissão e recepção do ultrassom respectivamente. Neste projeto os pinos GPIO 5 e GPIO 19 foram usados para o *echo*, e os

pinos GPIO 6 e GPIO 26 utilizados para o *trigger*. Um detalhe importante na instalação desse sensor é a criação de um circuito simples, pois o HC-SR04 trabalha com tensão 5V, enquanto o Raspberry Pi em 3.3V, logo, para evitar danos aos componentes, utilizou-se dois tipos de resistores, 1kΩ e 1.8kΩ, para criar um divisor de tensão, bastando ligar as duas pernas do resistor de 1kΩ ao echo, e o de 1.8kΩ com uma perna no GND e outra interligada ao de 1kΩ, como mostrado na Figura 17.

Os últimos sensores são os dois infravermelhos TCRT 5000. Pode notar-se pela Figura 17 que os dois sensores possuem um led infravermelho (nível superior) com ânodo e cátodo, e um fototransistor (nível inferior) com coletor e emissor. O led infravermelho é alimentado no ânodo por um resistor limitador de corrente de 330Ω e com o cátodo ligado ao GND, enquanto no fototransistor, o emissor é ligado ao GND e possui um resistor de 10kΩ no coletor, esse resistor que garante um sinal *HIGH* quando não há incidência de luz infravermelha, além de impedir um curto-circuito quando houver incidência. Para fazer a leitura do sinal são conectados os coletores dos sensores aos pinos GPIO 20 e GPIO 21, conforme Figura 2.



(a) (b)

Figura 18 – (a) Lateral do drone e (b) Parte de cima do drone

Fonte: Autoria própria

Apesar da Figura 17 mostrar os dois tipos de sensores instalados diretamente na BreadBoard, isso foi meramente para uma ilustração prática, pois esses sensores foram colocados em locais que devem realmente estar. Os sensores ultrassônicos foram colocados na parte frontal, como se fossem “para-choques”, enquanto os sensores infravermelhos foram colocados embaixo do drone, na parte da frente, um de cada lado, para que quando um detectar a presença da linha corrigir o trajeto. O resultado final do protótipo pode ser observado na Figura 18.

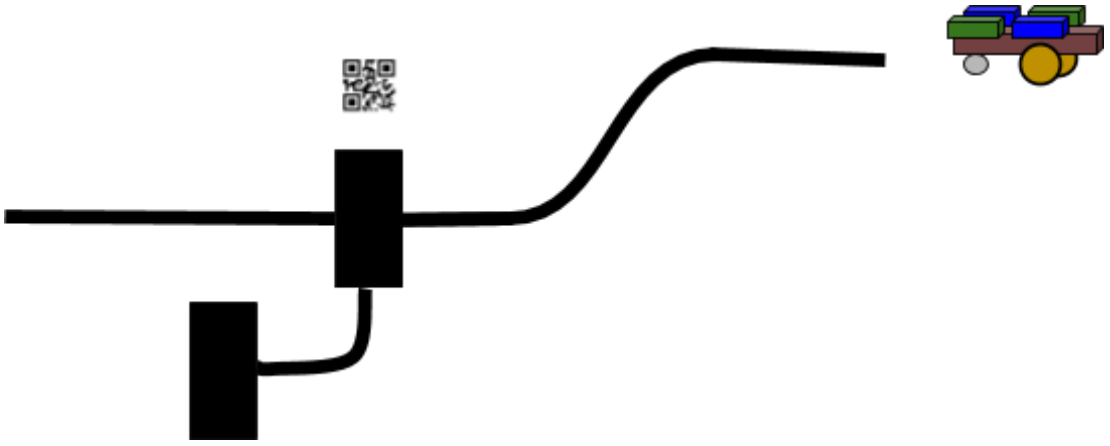


Figura 19 – Exemplo de pista para funcionamento do drone

Fonte: Autoria própria

O último elemento a ser feito nessa etapa, externo ao drone, é a montagem da pista. A pista deve ser composta por uma fita de qualquer cor, mas que tenha contraste ao chão, dessa forma, os sensores infravermelhos conseguem identificá-la. Como a Figura 19 mostra, curvas não podem ser retas (90°) para melhor eficácia dos sensores, e as estações devem ser demarcadas de ambos os lados com a mesma fita, assim, os sensores sabem onde está a estação para a parada. Para a identificação de cada estação, também é necessária a colocação dos QR codes impressos, esses que podem ser gerados e baixados no sistema web de controle. O sistema web para empresas deverá conter uma página exclusiva para instruções de montagem do trajeto e drone.

4.3.2 Montagem do Software

Acompanhando a montagem física foi-se desenvolvendo os códigos necessários para os testes, e que posteriormente, foram aproveitados no sistema drone. Seguindo a mesma

ordem da montagem física, começou-se pela implementação do controle dos motores, para isso, os pinos GPIO usados para o motor devem ser configurados como pinos de saída primeiramente. Dentro do mesmo arquivo há diversas funções responsáveis pelo movimento para frente, direita, esquerda, trás, e parada de emergência.

Com os motores prontos, foram programadas as funções para os sensores ultrassônicos então, dessa vez indicando os pinos GPIO de *trigger* (saída) e os pinos de *echo* (entrada). Logo após, foram implementadas algumas funções, a principal delas responsável por executar outras duas em thread e esperá-las finalizar antes de retornar um resultado. Estas duas funções que rodam em paralelo são medições para cada um dos sensores, os *triggers* emitem um sinal e esperam um tempo antes de finalizá-lo, em seguida, pega-se o horário do início da emissão e com os *echos* pega-se o horário do recebimento, faz-se uma operação de subtração para saber o tempo de duração do pulso, finalizando ainda com uma divisão por dois e uma multiplicação do mesmo por um valor correspondente a velocidade do som em determinada temperatura, com isso tudo obtêm-se a distância.

Outro arquivo importante para a interação do drone com o meio externo foi o de análise de frames capturados pela câmera para detecção e leitura de QR codes. Há apenas uma função, e ela recebe um frame e um *Id* de estação como parâmetros. Esta função faz alguns ajustes no frame e passa-o por uma função de detecção da biblioteca ZBAR, caso exista algum QR code no frame, verifica-se se o conteúdo (*Id* de uma estação) é igual ao *Id* recebido por parâmetro, e dessa forma, caso sejam iguais, é retornado *True*, senão é retornado *False*.

O drone já pode se movimentar, calcular distâncias e ler QR codes, porém, falta integrar todas essas operações para que ocorram em conjunto, por essa razão, há de ser implementado um arquivo de central de controle. Antes de tudo, é no centro de controle que se faz o *setup* de todos os pinos GPIO mencionados até o momento, incluindo os pinos GPIO dos sensores infravermelhos, nessa mesma função de *setup*, os sensores infravermelhos realizam uma leitura do sinal (*High* ou *Low*) recebido do chão e determinam-o como sinal padrão.

Na central de controle há a função principal que interliga todos os funcionamentos, como dito anteriormente, e a primeira ação é invocar como thread uma função responsável pelo constante controle da distância a obstáculos, se a distância entre um dos sensores e qualquer outro objeto for menor que 30 centímetros, é chamada a função de parada de emergência e bloqueia-se o uso dos motores até remoção do obstáculo. Enquanto os motores

estão livres, a câmera do Raspberry Pi captura frames, frames estes que são enviados a função de detecção de QR codes, caso seja retornado um True desta função, então é feito o processo de estacionamento, esperando que ambos os sensores infravermelhos detectem o sinal oposto ao padrão, para assim chegar a estação. De outro modo, pode não ser detectado alguma estação, ou a estação correta, consequentemente, ele deve continuar percorrendo o caminho até que ache a estação, este trajeto utiliza a constante leitura dos sensores infravermelhos, enquanto o sinal padrão se mantém em ambos os sensores, ele se movimenta para frente, mas a partir do momento que detectar uma mudança de sinal na esquerda, chama a função para virar a esquerda, e uma mudança de sinal na direita, chama a função para virar a direita, até que ambos voltem ao sinal padrão.

Outro arquivo tão importante quanto o centro de controle é o que possui o cliente do protocolo TCP, sem ele o drone fica sem conexão com o resto do sistema, e portanto, sem objetivo. Ele é o primeiro a ser executado no drone, e nele será invocado a central de controle. A primeira operação é conferir se já existe um “Raspberry” salvo localmente, se não houver, significa que é a primeira vez do drone executando, logo, ele inicia um protocolo para achar uma estação qualquer, e dessa forma, pode se conectar com o servidor e enviar a estação, o servidor por sua vez, criará um “Raspberry” para a mesma empresa que possui a estação enviada, esse “Raspberry” é retornado e salvo localmente para futuras inicializações. Também pode ocorrer de já existir um “Raspberry”, porém, desta vez é enviado o “Raspberry” para o servidor, que confere se há alguma alteração feita pela empresa no mesmo, e é retornado o “Raspberry” atualizado para ser salvo localmente. Ao longo de todo cliente ocorrem trocas de informações de mesmo modo, como a solicitação pela lista atualizada das estações para serem salvas localmente. O único processo onde é o servidor quem envia primeiro a solicitação é no caso de um “Comando”, este “Comando” é recebido e salvo localmente para que se comece a executá-lo, e só é retornado quando chegar a alguma das estações (inicial ou final) do “Comando”.

As três entidades citadas, “Raspberry”, “Estacao” e “Comando”, receberam classes próprias para auxiliar no processo do cliente, mas não mantiveram os mesmos atributos das entidades no banco de dados, somente os que são necessários para o funcionamento básico do drone. As três classes foram instanciadas num mesmo arquivo, e nesse arquivo utiliza-se o SQLAlchemy para a criação das tabelas. Outros três arquivos foram criados com as funções

de consulta e inserção de cada uma das três tabelas, eles também utilizaram o SQLALchemy para o diálogo com o banco de dados.

5.3.2 Conclusão

Após alguns poucos meses de desenvolvimento conseguiu-se obter excelentes resultados até o momento de escrita deste trabalho, como uma aplicação Android funcional que roda em versão mínima 5.0, e também quase todo o *WebService*, ambas conseguindo se comunicar sem problemas. A única parte das três que ainda não está operando como planejado é a parte do drone, mas será continuado seu desenvolvimento buscando que todo sistema consiga trabalhar em conjunto, nem que seja de forma básica. Este trabalho conseguiu dar um passo adentro deste novo modelo de organização conhecido como Indústria 4.0, trazendo para o conhecimento do leitor ferramentas simples mas que podem ser eficientes para grandes projetos referentes ou não a esse conceito.

Inclusive, com o *mobile* e o *WebService* prontos, esses ficarão a disposição de qualquer um que gostaria de utilizá-los para conectar seus drones a um sistema de controle, bastando utilizar o código base de operação do drone, que também será disponibilizado. Não se obteve apenas resultados ao projeto, mas também resultados pessoais, aprofundando conhecimentos já vistos no curso Técnico em Informática Integrado ao Ensino Médio ao longo de seus quatro anos, e a aprendizagem de novas tecnologias que foram necessárias para este projeto.

Contudo, um grande empecilho no início desse trabalho foi o conhecimento superficial de tecnologias que não contemplariam o suficiente para desenvolvê-lo, por essa razão, foram necessárias diversas leituras de documentações e fóruns referentes a maior parte do que foi apresentado. Tanto o Android quanto o Raspberry Pi, apesar de serem tecnologias conhecidas, tiveram que ser estudadas e compreendidas, de mesmo modo o uso do Spring no *WebService*, que não era uma tecnologia conhecida. Com tudo isso, surge como problema o tempo, desde o começo do trabalho até o momento de escrita desse, foram apenas 4 meses, estes que foram utilizados para a aprendizagem de muitas tecnologias e a implementação das mesmas, e isso sendo apenas em uma pessoa.

Outra dificuldade encontrada no desenvolvimento do sistema, foi a elaboração do fluxo do comando, que deveria funcionar de forma conjunta entre o *WebService* e o Android.

Atualmente, após alguns testes, o fluxo de comandos se apresentou eficaz e sem erros. De forma geral, a comunicação entre o *WebService* e o Android foi algo que apresentava constantes problemas, ainda mais nos casos onde o primeiro enviava mensagens sem requisições do segundo.

Há também as dificuldades que foram encontradas e não foram solucionadas, e que neste caso, podem ser muito úteis para futuros trabalhos. O primeiro problema para solucionar é a organização e a troca de informações entre o drone e o *WebService* pelo protocolo TCP/IP de forma funcional, arrumando a estrutura de serviço que atualmente está implementada, na qual não há um funcionamento como o esperado. Outro obstáculo com o drone é que, mesmo que estivesse pronto, não saberia fazer o menor caminho possível, até porque isso leva a um outro problema que é o uso de vários drones em uma mesma linha de estações, e essa situação não pode resultar em caos. Por fim, algo importante para um trabalho que venha a complementar esse, é a implementação de um sistema web para as empresas gerenciarem seus drones, estações e funcionários, segundo a Figura 4.

Portanto, com o presente trabalho, pode-se utilizá-lo como base para aperfeiçoamentos do mesmo, como foram mencionadas algumas ideias, mas não somente isso, podendo utilizá-lo, até mesmo, para a realização de trabalhos diferentes.

6 Referências

AFONSO, Alexandre. **O que é Spring Boot?** 2017. Disponível em: <https://blog.algaworks.com/spring-boot/>. Acesso em: 21 out. 2018

Andrade, Alexandre & Tiemi Soma, Andrea & Eiki, Cassio. (2016). **Automação de baixo custo baseada no Raspberry Pi**. 10.13140/RG.2.1.1282.3920. Disponível em: <https://www.researchgate.net/publication/303544027_Automacao_de_baixo_custo_baseada_no_Raspberry_Pi>. Acesso em: 16 out. 2018.

ANDROID STUDIO. **Conheça o Android Studio.** 2018. Disponível em: <https://developer.android.com/studio/intro/?hl=pt-br>. Acesso em: 19 out. 2018.

ANDROID STUDIO. **Manifesto do aplicativo.** 2018. Disponível em: <https://developer.android.com/guide/topics/manifest/manifest-intro?hl=pt-br>. Acesso em: 21 out. 2018.

CANDIDO, Gradimilo. **ROBÔ SEGUIDOR DE LINHA COM SENSOR INFRAVERMELHO E PWM.** 2018. Disponível em: <https://portal.vidadesilicio.com.br/robo-seguidor-de-linha-sensor-infravermelho-e-pwm/>. Acesso em: 01 out. 2018.

CARMO, Heitor Vital do. **Google Cloud Messaging: Introdução.** 2013. Disponível em: <https://www.devmedia.com.br/google-cloud-messaging-introducao/29776>. Acesso em: 20 set. 2018.

CIPOLI, Pedro. **Saiba tudo sobre o Raspberry Pi 3 e o que ele representa para o mercado.** 2016. Disponível em: <https://canaltech.com.br/hardware/saiba-tudo-sobre-o-raspberry-pi-3-59065/>. Acesso em: 15 out. 2018.

FIESP. **FIESP IDENTIFICA DESAFIOS DA INDÚSTRIA 4.0 NO BRASIL E APRESENTA PROPOSTAS.** 2018. Disponível em: <http://www.fiesp.com.br/noticias/fiesp-identifica-desafios-da-industria-4-0-no-brasil-e-apresenta-propostas/>. Acesso em: 07 out. 2018.

FLAUSINO, Rodrigo. **Eclipse.** 2007. Disponível em: <https://www.selectgame.com.br/eclipse/>. Acesso em: 05 ago. 2018.

G1. **Android passa Windows e se torna o sistema operacional mais usado do mundo.** 2017. Disponível em: <https://g1.globo.com/tecnologia/noticia/android-passa-windows-e-se-torna-o-sistema-operacional-mais-usado-do-mundo.ghtml>. Acesso em: 02 out. 2018.

GANTAN, Xiaonuo. **Introductory Tutorial of Python's SQLAlchemy.** 2013. Disponível em: <https://www.pythoncentral.io/introductory-tutorial-python-sqlalchemy/ >. Acesso em: 11 out. 2018.

GKATZIOURAS, Emmanouil. **Spring and Threads: TaskExecutor:** When you're working on long-running tasks for a web app, don't forget about Spring's TaskExecutor to. 2017. Disponível em: <https://dzone.com/articles/spring-and-threads-taskexecutor>. Acesso em: 28 set. 2018.

GOOGLE CHART. **QR Codes.** 2016. Disponível em: <https://developers.google.com/chart/infographics/docs/qr_codes>. Acesso em: 27 set. 2018.

HOLLER, Wilson. **Artigo: aplicações civis e comerciais de drones para os próximos anos.** 2018. Disponível em: <https://droneshowla.com/artigo-aplicacoes-civis-e-comerciais-de-drones-para-os-proximos-anos/>. Acesso em: 15 out. 2018.

MAVEN REPOSITORY. **Spring Boot Data JPA Starter.** Disponível em: <https://mvnrepository.com/artifact/org.springframework.boot/spring-boot-starter-data-jpa>. Acesso em: 11 set. 2018.

MAVEN REPOSITORY. **Spring Boot Web Starter.** Disponível em: <https://mvnrepository.com/artifact/org.springframework.boot/spring-boot-starter-web>. Acesso em: 11 set. 2018.

MEDEIROS, Higor. **Introdução à JPA - Java Persistence API.** 2013. Disponível em: <https://www.devmedia.com.br/introducao-a-jpa-java-persistence-api/28173>. Acesso em: 17 out. 2018.

MELO, Alexandre. **Mercado de tecnologia movimentou R$ 467,8 bilhões no Brasil em 2017.** 2018. Disponível em: <https://www.valor.com.br/empresas/5470331/mercado-de-tecnologia-movimentou-r-4678-bilhoes-no-brasil-em-2017>. Acesso em: 28 set. 2018.

MESQUITA, Arioto. O avanço dos drones: Criados com objetivos militares, os Vants – Veículos Aéreos Não Tripulados vêm sendo utilizados no Brasil por agroindústrias, empresas e, recentemente, fazendas, como instrumento de monitoramento e análise das lavouras.. **Os Drones Chegaram**, [s.i], p.20-25, maio 2014. Disponível em: <https://issuu.com/eriklm/docs/agro_ed_55_0b6439d9f921af>. Acesso em: 16 out. 2018.

NORWIED. **How to connect a python client to java server with TCP sockets.** 2012. Disponível em: <https://norwied.wordpress.com/2012/04/17/how-to-connect-a-python-client-to-java-server-with-tcp-sockets/>. Acesso em: 14 out. 2018.

O GLOBO. **Número de usuários únicos de celular chega a cinco bilhões no mundo.** 2018. Disponível em: <https://oglobo.globo.com/economia/numero-de-usuarios-unicos-de-celular-chega-cinco-bilhoes-no-mundo-22436866>. Acesso em: 08 out. 2018.

ONU BRASIL. **Brasil 'vai ser atropelado' por revolução digital e automação, avalia especialista.** 2018. Disponível em: <https://nacoesunidas.org/brasil-vai-ser-atropelado-por-revolucao-digital-e-automacao-avalia-especialista/amp/>. Acesso em: 28 set. 2018.

PYSCIENCE BRASIL. **Python: O que é? Por que usar?** Disponível em: <http://pyscience-brasil.wikidot.com/python:python-oq-e-pq>. Acesso em: 28 out. 2018.

RASPBERRY Pi Robotics #5: Line Follower. [s.i]: Explainingcomputers, 2017. (12 min.), son., color. Disponível em: <https://www.youtube.com/watch?v=Z5_8Va8QxnY>. Acesso em: 01 out. 2018.

ROBERTO, João. **O que é UUID? Porque usá-lo?** 2018. Disponível em: <https://medium.com/trainingcenter/o-que-%C3%A9-uuid-porque-us%C3%A1-lo-ad7a66644a2b>. Acesso em: 18 set. 2018.

ROSEBROCK, Adrian. **An OpenCV barcode and QR code scanner with ZBar.** 2018. Disponível em: <https://www.pyimagesearch.com/2018/05/21/an-opencv-barcode-and-qr-code-scanner-with-zbar/>. Acesso em: 16 out. 2018.

SIGAHI, Tiago Fonseca Albuquerque Cavalcanti; ANDRADE, Bárbara Cristina de. **A Indústria 4.0 na perspectiva da Engenharia de Produção no Brasil: levantamento e síntese de trabalhos publicados em congressos nacionais**. Enegep 2017 - Encontro

Nacional de Engenharia de Produção, [s.l.], p.1-13, 15 nov. 2017. ENEGEP 2017 - Encontro Nacional de Engenharia de Produção. http://dx.doi.org/10.14488/enegep2017_tn_stp_247_428_31208. Disponível em: <http://www.abepro.org.br/biblioteca/TN_STP_247_428_31208.pdf>. Acesso em: 07 out. 2018.

SILVEIRA, Cristiano Bertulucci; LOPES, Guilherme Cano. **O Que é Indústria 4.0 e Como Ela Vai Impactar o Mundo.**Disponível em: <https://www.citisystems.com.br/industria-4-0/>. Acesso em: 07 out. 2018.

SOUZA, Renato. **Brasil tem 700 mil acidentes de trabalho por ano.** 2017. Disponível em: <https://www.em.com.br/app/noticia/economia/2017/06/05/internas_economia,874113/brasil-tem-700-mil-acidentes-de-trabalho-por-ano.shtml>. Acesso em: 27 set. 2018.

SPRING. **Spring Boot.** Disponível em: <http://spring.io/projects/spring-boot>. Acesso em: 18 out. 2018.

SPRING. **SPRING INITIALIZR.** Disponível em: <https://start.spring.io/>. Acesso em: 25 ago. 2018.

SQLALCHEMY. **The Python SQL Toolkit and Object Relational Mapper.** Disponível em: <https://www.sqlalchemy.org/>. Acesso em: 11 out. 2018.

SUBRAMANIAN. **IR Sensor-TCRT5000 Pin And Working Details.** 2018. Disponível em: <https://www.androiderode.com/ir-sensor-tcrt5000-pin-and-working-details/>. Acesso em: 06 out. 2018.

TCRT5000 - Reflective Optical Sensor. 2012. Disponível em: <https://blog.zerokol.com/2012/11/tcrt5000-reflective-optical-sensor.html>. Acesso em: 06 out. 2018.

THOMSEN, Adilson. **Como conectar o Sensor Ultrassônico HC-SR04 ao Arduino.** 2011. Disponível em: <https://www.filipeflop.com/blog/sensor-ultrassonico-hc-sr04-ao-arduino/>. Acesso em: 04 out. 2018.

WOOD JUNIOR, Thomaz. **FORDISMO, TOYOTISMO E VOLVISMO: OS CAMINHOS DA INDÚSTRIA EM BUSCA DO TEMPO PERDIDO.** 1992. 13 f. Dissertação (Mestrado) - Curso de Administração de Empresa, Eaesp/fgv, São Paulo, 1992. Disponível em: <http://www.scielo.br/pdf/rae/v32n4/a02v32n4.pdf>. Acesso em: 28 set. 2018.