Bruno Ogata Franchi, RA:101893, Carolina Locatelli Colla, RA: 103219, Jaime Cazuhiro Ossada, RA:101989, João Victor de Mesquita Cândido dos Santos, RA:102028, Pedro Manhez Naresi, RA: 105615, Raphael Ribeiro Faria, RA:104120

Projeto R.USSIA - Aplicativo Monitorador de Filas do Restaurante Universitário do ICT-UNIFESP

São José dos Campos - Brasil 🇷🇺

Julho de 2018

Bruno Ogata Franchi, RA:101893, Carolina Locatelli Colla, RA: 103219, Jaime Cazuhiro Ossada, RA:101989, João Victor de Mesquita Cândido dos Santos, RA:102028, Pedro Manhez Naresi, RA: 105615, Raphael Ribeiro Faria, RA:104120

Projeto R.USSIA - Aplicativo Monitorador de Filas do Restaurante Universitário do ICT-UNIFESP

Relatório apresentado à Universidade Federal de São Paulo como parte dos requisitos para aprovação na disciplina de Projetos em Engenharia da Computação

Docente: Prof. Dr. Tiago de Oliveira

Universidade Federal de São Paulo - UNIFESP

Instituto de Ciência e Tecnologia - Campus São José dos Campos

São José dos Campos - Brasil 🇷🇺

Julho de 2018

Resumo

O relatório em questão visa apresentar o projeto desenvolvido ao longo da disciplina de projetos em engenharia da computação, sendo ele um aplicativo de monitoramento de fila do restaurante universitário do ICT- UNIFESP. Ao longo deste documento serão expostos detalhes sobre o planejamento do trabalho que visou diminuir o problema do tempo de espera excessivo que os alunos passam na fila do restaurante universitário, além de também citar o planejamento e a execução do projeto que utilizou o *hardware Raspberry Pi* para tirar fotos e um servidor web para fazer o processamento da imagens e inferir um *status* para a atual situação da fila. Tal resultado foi programado para ser mostrado em um aplicativo desenvolvido na plataforma *RShiny*.

Palavras-chaves: restaurante universitário. fila. processamento de imagem. aplicativo.

Lista de ilustrações

Sumário

1 Introdução

Presente em muitas universidades públicas do país, os restaurantes universitários tem total relevância na vida dos estudantes que frequentam suas faculdades diariamente, se apresentando como uma excelente alternativa para as refeições devido ao seu convidativo preço e sua facilidade de acesso.

Os preços dos restaurantes variam entre as universidades do Brasil, podendo variar entre 0,80 centavos a 3,00 reais dependendo da região. Considerando o fato de que grande parte dos estudantes de faculdades públicas não são nativos do local, tendo que sair de casa e se manter na cidade de sua universidade para continuar os estudos, o preço é um determinante vital para a alimentação no cotidiano.

No Restaurante Universitário (R.U) da Universidade Federal de São Paulo (Unifesp) no Campus São José dos Campos o preço é de 2,50 reais por refeição, para que o mesmo continue baixo para o benefício dos estudantes, é realizado um subsídio à empresa de forma a completar o valor integral do prato, o que é feito através de verbas federais. Além do valor baixo para consumo, os restaurantes universitários também apresentam uma qualidade aceitável dos alimentos, além de opções vegetarianas e uma fácil acessibilidade, visto que o restaurante se encontra na própria universidade, o que evita a necessidade de deslocamento dos alunos durante o período de refeição.

Nesta universidade, um problema muito recorrente no R.U, que é observado e comentado pela maioria dos alunos, é a extensão das filas do restaurante, que em horários de pico acaba ocupando toda a extensão do seu corredor externo. Dessa forma, o grupo teve a ideia de projetar um aplicativo que, com a ajuda de uma câmera posicionada de maneira estratégica, visa monitorar a situação atual da fila do restaurante, informando se o mesmo encontra cheio ou vazio através do tamanho da fila, e que pode ser consultado pelos alunos com o objetivo otimizar o tempo dos mesmos para que possam decidir melhor o horário para realizar sua refeição.

2 Objetivos

2.1 Geral

Visando otimizar o tempo desperdiçado na fila do Restaurante Universitário foi desenvolvido um aplicativo que tem como foco monitorar estas filas para poder avisar ao usuário qual é o *status* atual da fila para ingressar no restaurante, sendo esse *status* variando entre cheia ou vazia, fazendo com que o aluno possa fazer outras atividades no tempo em que estaria esperando para fazer sua refeição.

2.2 Específico

Para alcançar o êxito do trabalho, se fez necessário dividir a execução em etapas

- Pesquisa e compra dos componentes utilizados;
- Estudo da infraestrutura externa do Restaurante Universitário;
- Estudo de posicionamento e instalação de câmeras para análise;
- Estudo de como fazer a conexão entre o microcontrolador e a rede de *internet*;
- Desenvolvimento do sistema embarcado *hardware* a ser utilizado;
- Desenvolvimento do *software* para que seja realizada o processamento e análise de imagens para gerar um resultado
- Integração entre *hardware* e *software*;
- Entrega do produto final e funcional.

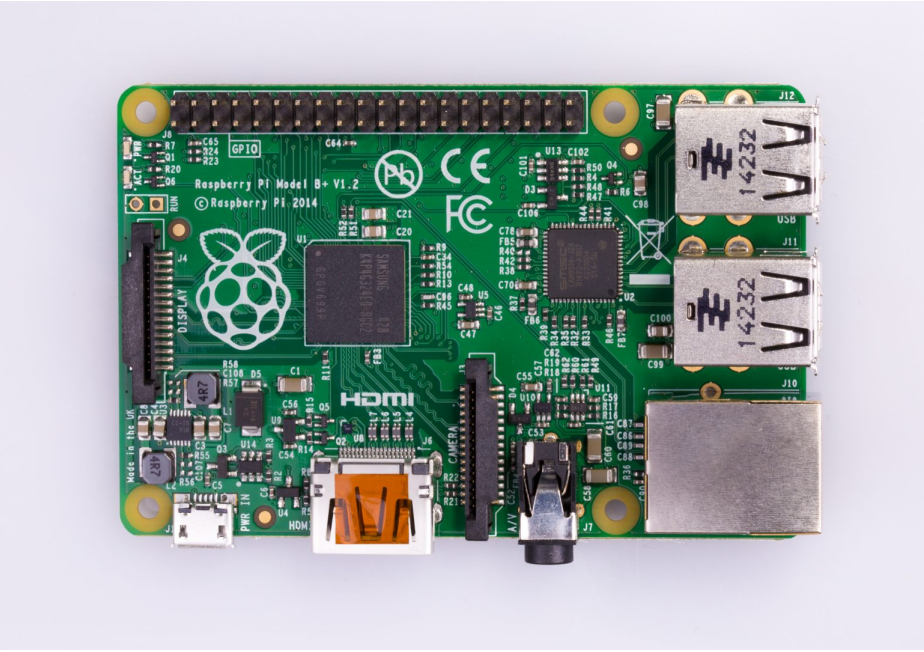
3 Fundamentação Teórica

3.1 *Raspberry Pi*

Raspberry Pi é um computador do tamanho de um cartão de crédito, e que cabe na palma da mão, se conectando a um monitor de computador ou TV, e usa um teclado e um mouse padrão, desenvolvido no Reino Unido pela Fundação *Raspberry* Pi. Todo o *hardware* é integrado numa única placa. O principal objetivo é promover o ensino em Ciência da Computação básica em escolas (1).

O Raspberry Pi 1 (utilizado no projeto) é baseado em um *System on a Chip* (SoC) Broadcom BCM2835, que inclui um processador ARM1176JZF-S de 700 MHz, GPU VideoCore IV, e 512 MB de memória RAM em sua última revisão. O projeto não inclui uma memória não-volátil - como um disco rígido - mas possui uma entrada de cartão SD para armazenamento de dados.

Figura 1 – Raspberry Pi 1



Fonte: Google

3.2 R-Shiny

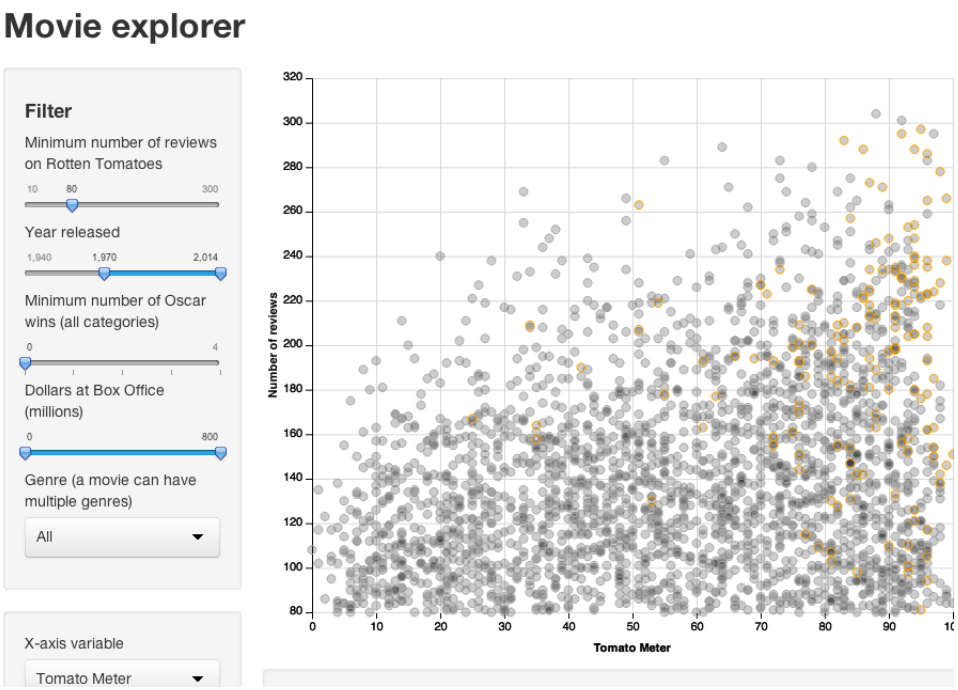
O Shiny é um sistema para desenvolvimento de aplicações *web* usando o R, um pacote do R (Shiny) e um servidor *web (Shiny Server)*. O *Shiny* é exatamente isso e nada

mais, portanto *Shiny* não é uma página web, não é um substituto para sistemas mais gerais, como*Ruby on Rails* e *Django*, e também não é uma ferramenta gerencial, como o *Tableau*.

O código de uma aplicação *Shiny* nos permite estruturar tanto a interface com o usuário quanto o processamento de dados, geração de visualizações e modelagem, isto é, nós programamos tanto o *user side* quanto o *server side* numa tacada só. Ao rodar o código, é criado um servidor que envia páginas *web*, recebe informações do usuário e processa os dados, utilizando apenas o R.

O pacote *shiny* do R possui internamente um servidor *web* básico, geralmente utilizado para aplicações locais, permitindo somente uma aplicação por vez. O *shiny server* é um programa que roda somente em Linux que permite o acesso a múltiplas aplicações simultaneamente (2).

Figura 2 – R-Shiny



Fonte: Google

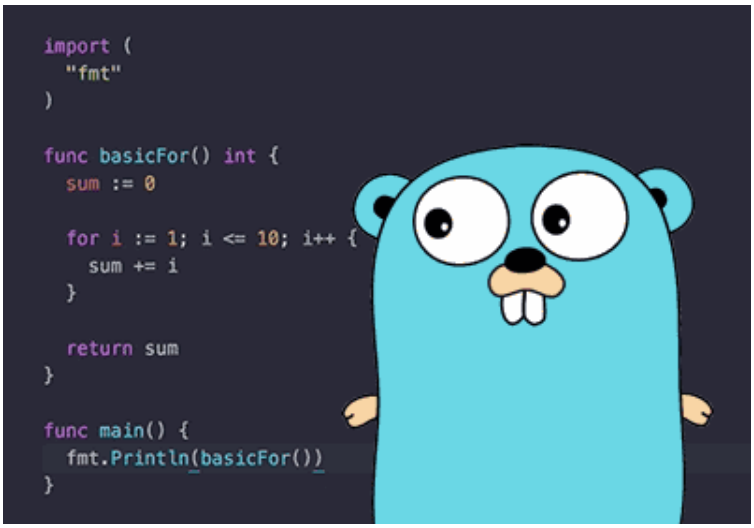
3.3 Go

Go é uma linguagem de programação criada pela *Google* e lançada em código livre em novembro de 2009. É uma linguagem compilada e focada em produtividade e programação concorrente, baseada em trabalhos feitos no sistema operacional chamado Inferno. O projeto inicial da linguagem foi feito em setembro de 2007 por *Robert Griesemer*,

Rob Pike e *Ken Thompson*. Atualmente, há implementações para *Windows*, *Linux*, Mac OS X e FreeBSD.

O Go vem sendo apontado como uma boa alternativa a linguagens com o *Ruby* e especialmente ao Java da *Oracle*. O Java foi desenvolvido há 20 anos pela *Sun Microsystems* e se tornou muito popular com o tempo, mas desenvolvedores tem reclamado constantemente da linguagem desde que a *Oracle* assumiu o controle, após ter adquirido a *Sun* em 2010, pela demora no lançamento de novas ferramentas de desenvolvimento (3).

Figura 3 – Go



Fonte: Google

3.4 Servidor

Em informática, um servidor é um software ou computador, com sistema de computação centralizada que fornece serviços a uma rede de computadores, chamada de cliente. Esses serviços podem ser de naturezas distintas, como por exemplo, arquivos e correio eletrônico. Esta arquitetura é chamada de modelo cliente-servidor, é utilizada em redes de médio e grande porte (com muitas máquinas) e em redes onde a questão da segurança desempenha um papel de grande importância.

O termo servidor é amplamente aplicado a computadores completos, embora um servidor possa equivaler a um *software* ou a partes de um sistema computacional, ou até mesmo a uma máquina que não seja necessariamente um computador.

Os servidores podem fornecer várias funcionalidades, muitas vezes chamado de "serviços", tais como a partilha de dados ou de recursos do sistema entre vários clientes, ou computação desempenho para um cliente. Um único servidor pode servir vários clientes, e um único cliente pode usar vários servidores. Um processo cliente pode ser executado

no mesmo dispositivo ou pode se conectar através de uma rede para um servidor em um dispositivo diferente (4).

Figura 4 – Servidor

Fonte: Google

4 Desenvolvimento

O projeto foi desenvolvido ao longo de etapas sendo as principais a constatação da lista de materiais que foram utilizados no projeto, o estudo da infraestrutura do RU e o desenvolvimento do sistema

4.1 Lista de Materiais

Para o projeto foram adquiridos:

- 1x - Raspberry Pi;
- 1x - Câmera de Raspberry Pi;
- 1x - Cabo de internet;

4.2 Estudo da infraestrutura do RU

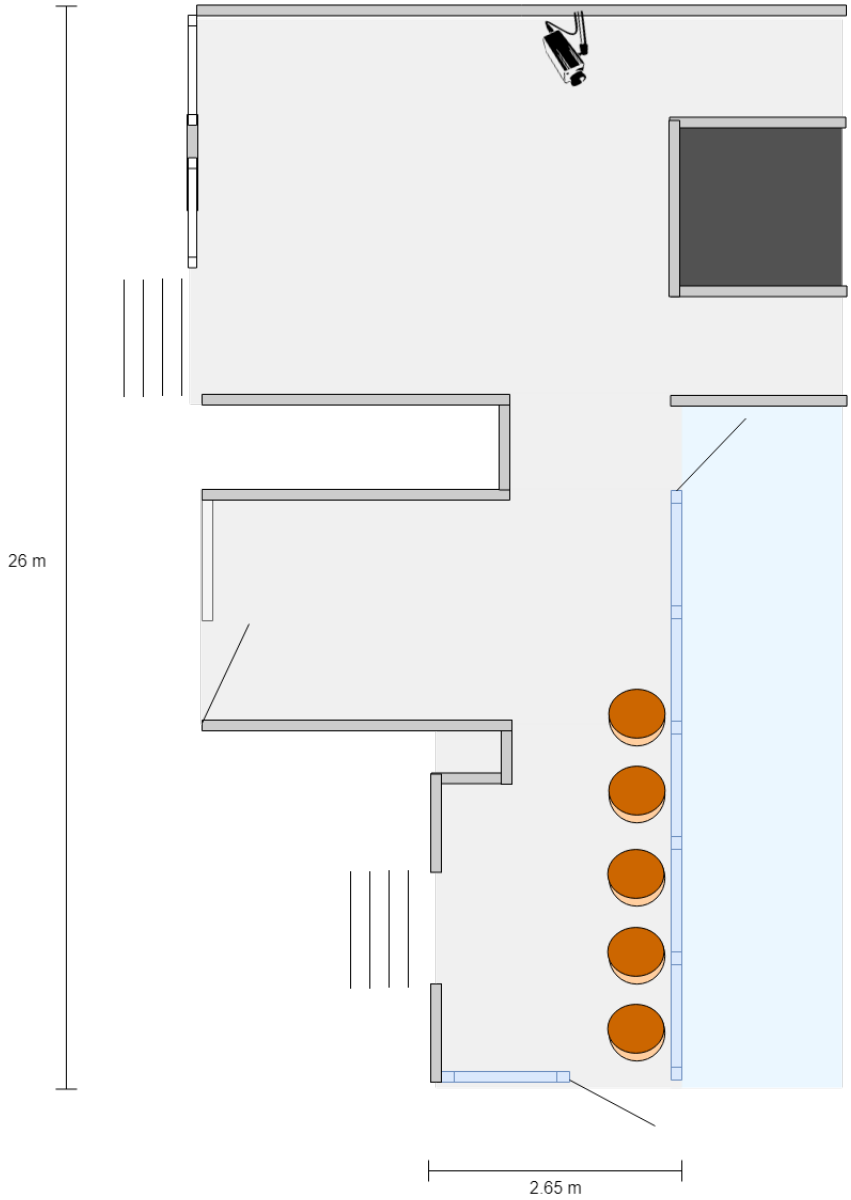
Como as filas do R.U. acabam preenchendo quase toda a extensão de seu corredor externo torna-se inviável a utilização de muitas câmeras para capturar seu comportamento em toda sua extensão. Pensando nisso, foi definido um ponto chave para a alocação da câmera para que assim se pudesse obter o resultado desejado.

A câmera seria situada no final do corredor, garantindo um campo de visão mais amplo para que as fotos pudessem ser tiradas abrangendo uma maior área como mostrado na Figura 5.

A escolha desta posição também foi influenciada pela possibilidade de falso positivo caso a foto tirada contivesse pessoas de dentro do R.U, o que poderia acontecer pelo fato de metade da parede ser composta por vidro (transparente) deixando que as pessoas de fora, na fila, fossem confundidas com pessoas que já tivessem ingressado dentro do R.U pelo processador de imagem. Esse fato ocasionaria um mau funcionamento no projeto fornecendo resultados imprecisos.

Este novo posicionamento da câmera escolhido pelo grupo para a continuação do projeto, utilizada com o *Raspberry*, contorna esse problema surgido e garante resultados melhores e mais precisos a cerca do *status* atual da fila do restaurante no momento.

Figura 5 – Estrutura do R.U.



Fonte: Autoria própria

4.2.1 Viabilidade de Internet

Durante a etapa de estudo da infraestrutura do RU foi feito uma reunião com o T.I da universidade a fim de verificar a viabilidade de obtenção da *internet* necessária para que o protótipo pudesse tirar fotos e enviar para serem processadas.

Foi constatado que a internet poderia ser obtida através de uma sala localizada dentro do R.U. onde já exite internet, sendo necessário apenas cabos ou um roteador para viabilizar a utilização da mesma.

Vale ressaltar que o protótipo criado pode ser utilizado para o monitoramento de diversos tipos de filas não ficando restrito a filas de restaurantes universitários.

4.3 Primeira Tentativa

O projeto passou por modificações desde sua concepção até sua finalização, sendo que a primeira tentativa de desenvolvimento do trabalho foi utilizar o microcontrolador Arduino Mega 2560 juntamente com 3 módulos de câmeras próprios para o sistema e resistores diversos, conectados através de uma *protoboard*.

Após as câmeras já terem sido compradas pelos integrantes, foi percebido um grave equívoco por parte do grupo, pelo fato de essas câmeras para Arduino não possuírem o circuito integrado chamado de FIFO (*First In First Out*), cuja função basicamente é empilhar as informações que foram recebidas, no caso as fotos que fossem tiradas ao longo do processo, e depois enviá-las na mesma ordem que entraram, já que o microcontrolador não pode realizar o download das imagens na mesma velocidade que a câmera envia os dados.

Esse fato aumentaria consideravelmente a complexidade do projeto pelo fato de que cada câmera necessitar também de um cartão de memória para armazenamento, acesso *Wi-Fi* independente e um Arduíno próprio para obter o suporte necessário aos seus recursos, o que tornaria o projeto inviável de ser implementado em um sistema real.

Por essas razões aqui apresentadas, foi decidido pelo grupo que modificar a ideia original de modo a contornar o problema seria a melhor alternativa para o projeto, optando pelo uso do microcomputador *Raspberry Pi* (Figura 6). Essa alternativa foi a solução que melhor se encaixou dentro da proposta do trabalho a ser desenvolvido, levando em conta o tempo disponível como um determinante importante, visto que primeira tentativa de execução ocasionou um atraso de cerca de um mês e meio no cronograma apresentado.

Figura 6 – Raspberry Pi e sua câmera

Fonte: Autoria própria

4.4 Nova abordagem

Após decidido a nova abordagem do projeto, e se aproveitando da ocasião de que um dos integrantes do grupo já dispunha de um *Raspberry Pi* versão 1, foi possível também se obter uma câmera própria para esse componente através de um conhecido. Pelo fato da amostra disponível ser uma versão antiga, foi encontrada uma nova dificuldade de projeto que se resumiu no fato de quando foi realizada a atualização do computador *Raspberry*, o seu sistema operacional atual se mostrava muito pesado, assim como a experiência de usuário que também necessitava de uma grande quantidade de memória.

Para a resolução desse impasse, a alternativa encontrada foi optar por um sistema operacional que funcione por meio de uma interação com terminal, fazendo com que, desta forma, a parte gráfica do projeto não adquira um tamanho pesado sem necessidade.

Quando se deu início a nova abordagem do projeto, uma outra dificuldade encontrada era sobre quais ferramentas utilizar para o desenvolvimento do aplicativo, e de início, foi utilizado um desenvolvedor chamado *AppyPie*, cujas ferramentas pareciam satisfazer as necessidades que o projeto precisava. Entretanto, este não conseguia fazer requisições em tabelas como gostaríamos, pois se comunica com o banco de dados *Firebase*, além de que muitas das funcionalidades necessárias para finalização do aplicativo necessitam de assinatura paga.

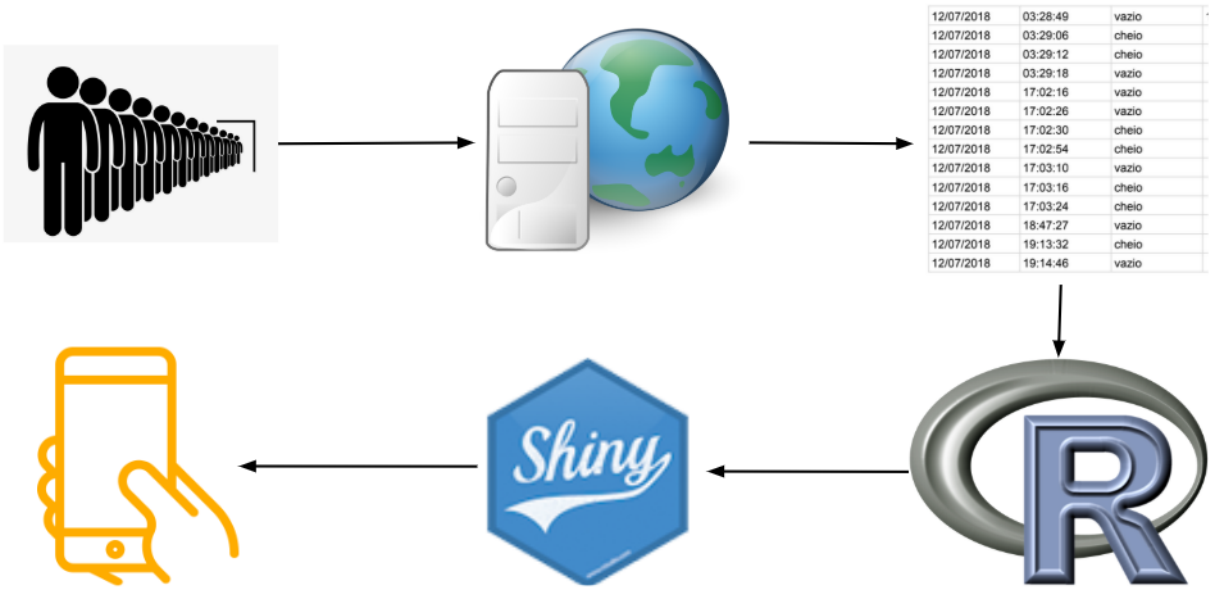
A segunda abordagem que foi pensada sobre o aplicativo era de utilizar o *framework Ionic*, que é muito útil quando se trata do desenvolvimento de aplicativos de alto padrão, utilizando três linguagens de programação que são o HTML5, para a parte visual do aplicativo, SCSS, para a parte de servidores e comunicação com banco de dados, e *Typescript*, que é utilizado para realizar toda a lógica por trás do aplicativo. Porém, o problema é que este *framework* também não conseguiria realizar as requisições de uma tabela, como era esperado para gerar análises estatísticas em tempo real, fazendo com que houvesse uma mudança estratégica que foi a de utilizar o R-*Shiny*, que consegue suprir todas as necessidades que o aplicativo monitorador de filas necessita.

4.4.1 Servidor

O Servidor Web foi desenvolvido utilizando Go e requisição HTTP, que é um método usado para enviar e receber informações advindas da *web*. Quando é feita uma requisição, faz-se necessário também identificar o método que será utilizado, identificando assim qual a ação que deverá ser tomada em um dado recurso (5).

Na Figura 7 se encontra um diagrama que ilustra todo o processo passo a passo que envolve o projeto aqui apresentado. A execução é iniciada no momento em que uma foto do é tirada via *Raspberry Pi* da fila atual do restaurante, e após isso, a mesma é enviada para o servidor web implementado que será responsável pelo processamento geral

Figura 7 – Diagrama do projeto



Fonte: Autoria própria

da imagem, encaminhando o resultado obtido para uma tabela do *Google Spreadsheet* construída para armazenar as informações recebidas do servidor, ilustrada na Figura 8.

Figura 8 – Tabela do Google Spreadsheet

12/07/2018	03:28:49	vazio
12/07/2018	03:29:06	cheio
12/07/2018	03:29:12	cheio
12/07/2018	03:29:18	vazio
12/07/2018	17:02:16	vazio
12/07/2018	17:02:26	vazio
12/07/2018	17:02:30	cheio
12/07/2018	17:02:54	cheio
12/07/2018	17:03:10	vazio
12/07/2018	17:03:16	cheio
12/07/2018	17:03:24	cheio
12/07/2018	18:47:27	vazio
12/07/2018	19:13:32	cheio
12/07/2018	19:14:46	vazio

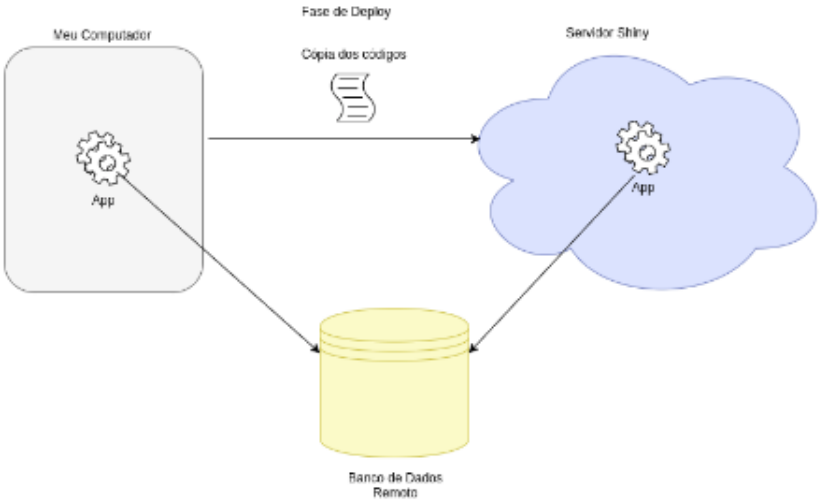
Fonte: Autoria própria

A tabela contém as informações pertinentes sobre a data da foto, a hora em que foi tirada e o estado em que a fila do R.U. se encontra no determinado momento, atualizando

o estado do restaurante a cada foto tirada.

Esses dados recebidos são processados no R, utilizando-se de um banco de dados remoto (*Google Spreadsheet)*, e expostos ao *Shiny* como mostrado na Figura 9, onde é feita uma cópia dos respectivos códigos e dados para o servidor *Shiny*, que assim que conclui todas as etapas, permite que o aplicativo, e consequentemente o usuário, possa consultar a atual situação do restaurante da universidade.

Figura 9 – Fase de Deploy



Fonte: Autoria própria

4.4.2 Processamento de Imagem

O processamento das imagens recebidas do R.U. para verificação do estado atual que a fila se encontra foi feito em cima do padrão de cores RGBA (*Red Green Blue Alpha*), que ao contrário do sistema mais comumente utilizado RGB, (6) tem um quarto canal que é responsável pela opacidade das cores, onde os valores de cores RGBA presente em cada *pixel* são analisados.

No processamento, existem dois tipos de imagem que são comparadas: a imagem Núcleo e a imagem Teste. A primeira imagem se trata de uma foto do fundo (parede) sem a presença da fila de alunos no restaurante, enquanto a imagem Teste se tratam de fotos que foram tiradas durante a execução do sistema embarcado desenvolvido.

Como os *pixels* de cada foto são formados por componentes de cores do padrão RGBA, é possível calcular a distância euclidiana entre cada pixel para ao fim se obter um erro médio da imagem Teste com a Núcleo, que irá determinar se a fila se encontra cheia ou vazia.

A fila assume o estado de vazia quando este erro médio calculado é de pequena escala, que ocorre devido a possíveis pequenas variações de iluminação no local, visto que

o mesmo não se encontra com muitas pessoas, e a fila é considerada cheia quando este erro médio é grande, pois um grande número de alunos no local analisado resulta em um número alto de variações ocorridas devido à presença de uma grande quantidade de cores diferentes provenientes das vestimentas, acessórios, cores de peles e outros fatores das pessoas que se encontram no local.

4.4.3 Aplicativo

Para o desenvolvimento do aplicativo, já se tinha em mente o uso de gráficos para inferir alguma estatística sobre a situação atual da fila, para isso foi utilizado o *R-Shiny*. O uso de tal ferramenta se fez essencial pois com ela foi possível analisar o que era necessário para o desenvolvimento de uma interface amigável para o usuário.

Assim, a partir do momento que era feita uma requisição na tabela do *SpreadSheet* era possível analisar a última célula desse banco de dados que já foi apresentado na Figura 8 verificando a informação que essa célula contem.

Quando a última célula do banco de dados é inserida, o aplicativo faz uma análise da terceira coluna para averiguar qual é a situação atual da fila (cheia ou vazia) e retorna esse *status* para o usuário por meio do próprio aplicativo mostrando essa informação para o usuário de maneira intuitiva.

Além disso, foi possível realizar uma análise temporal fazendo uma consulta de como estava a fila nos últimos cinco minutos, ou seja, pode-se verificar a cada momento o *status* da fila e com isso criar um gráfico para inferir uma análise estatística do tempo médio de espera na fila enquanto ela está cheia, auxiliando o aluno na escolha de se vale ir para a fila ou se seria melhor esperar mais tempo até ela ficar vazia.

Vale ressaltar que o R já vem sendo reconhecido pelo seu uso em estatística e em *bootstrap*, que se adapta tanto a *desktop* quanto a celular. Sendo assim essa ferramenta tem grande valor para inferências e análises agregando utilidade ao projeto trabalhando com os dados processados.

4.5 Divisão de tarefas

A divisão de tarefas foi feita dividindo a equipe em dois grupos, um responsável pelo desenvolvimento do aplicativo e o outro responsável pelo servidor web incluindo a parte do processamento da imagem.

Diversas reuniões foram feitas ao longo do semestre para que os membro dos grupos pudessem se atualizar sobre o que o outro grupo estava desenvolvendo e entender não somente uma parte mas sim o todo do trabalho.

O relato individual dos alunos que compõem o grupo encontram-se no Anexo 1.

4.5.1 Aplicativo

Os responsáveis pelo aplicativo foram os alunos João Victor de Mesquita, Bruno Ogata e Pedro Naresi. As reuniões foram de vital importância para esse grupo pelo fato de o aplicativo ter que ser desenvolvido de forma paralela, entre os integrantes que desenvolviam as diferentes partes do código, e ao mesmo tempo de forma integrada, para que pudesse haver a junção dos mesmos, o que era necessário periodicamente para verificar se o desenvolvimento do mesmo estava correto, não havendo conflitos no código integrado.

4.5.2 Servidor

O servidor responsável pelo processamento da foto foi feito pelos alunos Jaime Cazuhiro Ossada, Carolina Locatelli Colla e Raphael Ribeiro Faria. Eles ficaram responsáveis por utilizar o *Raspberry Pi* para tirar as fotos e criar um servidor web que processava as imagens tiradas.

Esse grupo, assim como o primeiro, realizou diversas reuniões sendo o código desenvolvido durante esses encontros e não de forma paralela como o primeiro. Neste ponto a conciliação de horários contribuiu para o bom desenvolvimento desta etapa.

4.5.3 Integração

A integração do projeto foi feito por todo o grupo, pois fez-se necessário o conhecimento de todas as partes dos códigos para que o aplicativo pudesse se comunicar de forma eficiente e correta com o servidor web.

5 Resultados Obtidos e Discussões

5.1 Códigos

5.1.1 Servidor

Assim que o servidor recebe as fotos, o algoritmo que é responsável pela identificação das imagens através do padrão RGBA, como já esclarecido anteriormente, foi implementado com várias funções para que seja possível realizar as tarefas desejadas. Primeiramente, as duas imagens que serão usadas no programa, núcleo e teste, são carregadas pela função *LoadImage()*, que retorna um código de erro caso ocorra algum impasse durante o carregamento.

Após as duas imagens carregadas, é necessário que os respectivos *pixels* sejam armazenados para que as comparações possam ser realizadas, e para isso, foi criado um tipo de *struct* com 4 variáveis de 32 *bits* para desempenhar essa função. A função *GetArrayPixelsRGBA()*, com o auxílio de um laço que vai de zero até o tamanho da imagem, lê os *pixels* desejados da imagem e armazena os mesmos no vetor criado. Em seguida, estando os dados já armazenados, a função *CalculateMeanDistance()* é chamada para, juntamente com uma função interna que calcula a distância euclidiana entre os *pixels*, obter o erro médio que irá determinar a situação observada na foto recebida.

Após efetuado todos os cálculos, o programa compara o resultado obtido com os valores definidos para cada situação. Caso o erro médio seja maior que 1000, o servidor retorna "cheio"como situação atual do restaurante, e caso contrário, retorna "vazio"e envia os dados correspondentes para a tabela, sendo que sua implementação pode ser observada no Algoritmo 1 abaixo.

Algoritmo 1 - Tratamento de imagens

```
1  package main
2
3  import(
4  //      "fmt"
5          "image"
6          "image/jpeg"
7          "os"
8          "log"
9          "math"
10 )
11
12 var pixels_base []pixel
13
14 type pixel struct {
15         r, g, b, a uint32
```

```
16 }
17
18 func init(){
19         image.RegisterFormat("jpeg", "jpeg", jpeg.Decode, jpeg.DecodeConfig)
20 }
21
22 func LoadImage() (image.Image, image.Image){
23         imgcore, err := os.Open("./nucleo.jpg")
24         imgdata, err2 := os.Open("./teste.jpg")
25
26         if (err != nil) || (err2 != nil) {
27                 log.Fatal(err)
28         }
29
30         base, err_ := jpeg.Decode(imgcore)
31         data, _ := jpeg.Decode(imgdata)
32
33         if err_ != nil {
34                 log.Fatal(err_)
35         }
36
37         return base, data
38 }
39
40 func GetArrayPixelsRGBA(pic image.Image, x_bound int, y_bound int) []pixel{
41         total_bound := x_bound * y_bound
42         pixarr := make([]pixel, total_bound)
43
44         for i := 0; i < total_bound; i++ {
45                 x := i % x_bound
46                 y := i / x_bound
47
48                 r, g, b, a := pic.At(x,y).RGBA()
49
50                 pixarr[i].r = r
51                 pixarr[i].g = g
52                 pixarr[i].b = b
53                 pixarr[i].a = a
54         }
55
56         return pixarr
57 }
58
59 func CalculateMeanDistance(pixarr1 []pixel, pixarr2 []pixel, x_bound int, y_bound int)
       float64{
60         deviation := float64(0)
61         total_bound := x_bound * y_bound
62
63         for i := 0; i < total_bound; i++ {
64                 p1_r, p1_g, p1_b, p1_a := pixarr1[i].GetRGBAValues()
65                 p2_r, p2_g, p2_b, p2_a := pixarr2[i].GetRGBAValues()
66                 dist := EuclidianDistance(p1_r, p1_g, p1_b, p1_a, p2_r, p2_g, p2_b, p2_a)
67                 deviation += dist
68         }
69
70         return (deviation / float64(total_bound))
71 }
72
73 /*func main(){
74
```

```
        base, data := LoadImage()

        fmt.Println(GetStateQueue(base, data))
        fmt.Println(GetStateQueue(nil, data))
}*/
func SquareDiff(x, y uint32) uint64 {
    d := uint64(x) - uint64(y)
    return d * d
}

func EuclidianDistance(a1 uint32,b1 uint32,c1 uint32,d1 uint32,a2 uint32,b2 uint32,c2
    uint32,d2 uint32) float64{
        dist := SquareDiff(a1,a2) + SquareDiff(b1,b2) + SquareDiff(c1,c2) + SquareDiff(d1
            ,d2)
        dist64 := float64(dist)

        return math.Sqrt(dist64)
}

func (p pixel) GetRGBAValues() (uint32,uint32,uint32,uint32){
        return p.r, p.g, p.b, p.a
}

func GetStateQueue(base image.Image, data image.Image) string{
        bounds := data.Bounds()

        if(base != nil){
                pixels_base = GetArrayPixelsRGBA(base, bounds.Dx(), bounds.Dy())
        }
        pixels_data := GetArrayPixelsRGBA(data, bounds.Dx(), bounds.Dy())

        mean_dist := CalculateMeanDistance(pixels_base, pixels_data, bounds.Dx(), bounds.
            Dy())

        if mean_dist > float64(1000) {
                return "cheio"
        } else {
                return "vazio"
        }
}
```

Fonte: Autoria própria

Para que o servidor possa receber a imagem do *Raspberry Py*, que é enviada de maneira *online*, é necessário que seja feita uma requisição HTTP (*Hypertext Transfer Protocol*), que se trata basicamente do método mais utilizado hoje em dia para enviar e receber informações da web. No algoritmo abaixo é ilustrado como o servidor criado realiza essa requisição, enviando as fotos do *raspberry* para o servidor em si, que se for concluída com sucesso, envia o arquivo para o local onde será realizada a cópia. Isso pode ser visto no Algoritmo 2 abaixo.

Algoritmo 2 - Envio de Imagem

```
package main

```

```
3  import (
4      "log"
5      "net/http"
6      "io"
7      "bytes"
8      "os"
9      "mime/multipart"
10     "encoding/json"
11 )
12
13
14 func MakeRequest() {
15
16         // Open the file
17         file, err := os.Open("teste.jpg")
18         if err != nil {
19                 log.Fatalln(err)
20         }
21         // Close the file later
22         defer file.Close()
23
24         // Buffer to store our request body as bytes
25         var requestBody bytes.Buffer
26
27         // Create a multipart writer
28         multiPartWriter := multipart.NewWriter(&requestBody)
29
30         // Initialize the file field
31         fileWriter, err := multiPartWriter.CreateFormFile("file_field", "teste.jpg")
32         if err != nil {
33                 log.Fatalln(err)
34         }
35
36         // Copy the actual file content to the field field's writer
37         _, err = io.Copy(fileWriter, file)
38         if err != nil {
39                 log.Fatalln(err)
40         }
41
42         // Populate other fields
43         fieldWriter, err := multiPartWriter.CreateFormField("normal_field")
44         if err != nil {
45                 log.Fatalln(err)
46         }
47
48         _, err = fieldWriter.Write([]byte("Value"))
49         if err != nil {
50                 log.Fatalln(err)
51         }
52
53         // We completed adding the file and the fields, let's close the multipart writer
54         // So it writes the ending boundary
55         multiPartWriter.Close()
56
57         // By now our original request body should have been populated, so let's just use
               it with our custom request
58         req, err := http.NewRequest("POST", "http://localhost:8080", &requestBody)
59         if err != nil {
60                 log.Fatalln(err)
61         }
```

```
62          // We need to set the content type from the writer, it includes necessary
                boundary as well
63          req.Header.Set("Content-Type", multiPartWriter.FormDataContentType())
64
65          // Do the request
66          client := &http.Client{}
67          response, err := client.Do(req)
68          if err != nil {
69                  log.Fatalln(err)
70          }
71
72          var result map[string]interface{}
73
74          json.NewDecoder(response.Body).Decode(&result)
75
76          log.Println(result)
77  }
78
79  func main(){
80          MakeRequest()
81  }
```

Fonte: Autoria própria

Após obtida a imagem, o servidor cria um arquivo e constrói uma cópia para realizar a escrita na tabela, passando como parâmetros para *quickstart.WriteString()*, que irá realizar esse próximo passo, a data da foto, o horário atual em que a foto foi tirada no restaurante, e o respectivo estado da fila no momento, ambos já estabelecidos e definidos anteriormente pelo Algoritmo 2 já explicado de tratamento de imagem.

Além disso, é definido e enviado também o respectivo formato desejado que as informações devem ser escritas nas tabelas, visando obter uma padronização de modo a facilitar o armazenamento e consulta destes dados como pode ser visto no Algoritmo 3 abaixo, que se refere à implementação do servidor em si.

Algoritmo 3 - Servidor de Arquivo

```
1   package main
2
3   import (
4       "fmt"
5       "log"
6       "net/http"
7       "io"
8     //  "bytes"
9       "strings"
10      "os"
11      "./quickstart"
12      "time"
13  )
14
15  func loadFile(w http.ResponseWriter, r *http.Request) {
16          //var Buf bytes.Buffer
17      // in your case file would be fileupload
18      file, header, err := r.FormFile("file_field")
```

```
19      if err != nil {
20          panic(err)
21      }
22      defer file.Close()
23      name := strings.Split(header.Filename, ".")
24      fmt.Printf("File name %s\n", name[0])
25
26      fmt.Printf("Creating new file\n")
27          out, err := os.Create("./teste.jpg")
28          if err != nil {
29                  fmt.Printf("Unable to create the file for writing. Check your write
                        access privilege\n")
30                  return
31          }
32
33          defer out.Close()
34
35          // write the content from POST to the file
36          byCp, errr := io.Copy(out, file)
37          fmt.Printf("Copied " +  string(byCp) + "\n");
38          if errr != nil {
39                  fmt.Fprintln(w, errr)
40          }
41
42          fmt.Printf("File uploaded successfully : \n")
43          fmt.Printf(header.Filename + "\n")
44      return
45  }
46  func loadHandler(w http.ResponseWriter, r *http.Request) {
47      loadFile(w , r)
48
49      base, data := LoadImage()
50      //fmt.Println(GetStateQueue(base, data))
51      currentTime := time.Now()
52      quickstart.WriteString(currentTime.Format("02/01/2006"), currentTime.Format("15:04:05
            "), GetStateQueue(base, data));
53  }
54
55  func main() {
56      http.HandleFunc("/", loadHandler)
57      log.Fatal(http.ListenAndServe(":8080", nil))
58  }
```

Fonte: Autoria própria

O Algoritmo 4 abaixo corresponde ao arquivo *quickstart* realizará o passo final de enviar e escrever os dados obtidos pelo servidor para a uma tabela do *Google Spreadsheets*, que funcionará como uma espécie de banco de dados para que o aplicativo desenvolvido possa realizar a consulta da situação atual da fila, assim como o respectivo horário e data em que aquele estado foi definido.

Após confirmar o sucesso do recebimento do *token* da *web*, o algoritmo salva e decodifica esse conteúdo recebido, retornando sempre um código de erro caso alguma etapa não se cumpra, e assim que necessário, com o endereço correspondente da tabela criada, realiza a escrita dos dados desejados para que o aplicativo possa realizar a consulta, como

ilustrado na Figura 8.

Algoritmo 4 - Quickstart

```
package quickstart

import (
        "encoding/json"
        "fmt"
        "io/ioutil"
        "log"
        "net/http"
        "os"

        "golang.org/x/net/context"
        "golang.org/x/oauth2"
        "golang.org/x/oauth2/google"
        "google.golang.org/api/sheets/v4"

        "strconv"
)

// Retrieve a token, saves the token, then returns the generated client.
func getClient(config *oauth2.Config) *http.Client {
        tokFile := "token.json"
        tok, err := tokenFromFile(tokFile)
        if err != nil {
                tok = getTokenFromWeb(config)
                saveToken(tokFile, tok)
        }
        return config.Client(context.Background(), tok)
}

// Request a token from the web, then returns the retrieved token.
func getTokenFromWeb(config *oauth2.Config) *oauth2.Token {
        authURL := config.AuthCodeURL("state-token", oauth2.AccessTypeOffline)
        fmt.Printf("Go to the following link in your browser then type the "+
                "authorization code: \n%v\n", authURL)

        var authCode string
        if _, err := fmt.Scan(&authCode); err != nil {
                log.Fatalf("Unable to read authorization code: %v", err)
        }

        tok, err := config.Exchange(oauth2.NoContext, authCode)
        if err != nil {
                log.Fatalf("Unable to retrieve token from web: %v", err)
        }
        return tok
}

// Retrieves a token from a local file.
func tokenFromFile(file string) (*oauth2.Token, error) {
        f, err := os.Open(file)
        defer f.Close()
        if err != nil {
                return nil, err
        }
        tok := &oauth2.Token{}
```

```
56          err = json.NewDecoder(f).Decode(tok)
57          return tok, err
58  }
59
60  // Saves a token to a file path.
61  func saveToken(path string, token *oauth2.Token) {
62          fmt.Printf("Saving credential file to: %s\n", path)
63          f, err := os.OpenFile(path, os.O_RDWR|os.O_CREATE|os.O_TRUNC, 0600)
64          defer f.Close()
65          if err != nil {
66                  log.Fatalf("Unable to cache oauth token: %v", err)
67          }
68          json.NewEncoder(f).Encode(token)
69  }
70
71  func WriteString(date string, hour string, state string) {
72          b, err := ioutil.ReadFile("client_secret.json")
73          if err != nil {
74                  log.Fatalf("Unable to read client secret file: %v", err)
75          }
76
77          // If modifying these scopes, delete your previously saved client_secret.json.
78          config, err := google.ConfigFromJSON(b, "https://www.googleapis.com/auth/
                spreadsheets")
79          if err != nil {
80                  log.Fatalf("Unable to parse client secret file to config: %v", err)
81          }
82          client := getClient(config)
83
84          srv, err := sheets.New(client)
85          if err != nil {
86                  log.Fatalf("Unable to retrieve Sheets client: %v", err)
87          }
88
89          // Prints the names and majors of students in a sample spreadsheet:
90          // https://docs.google.com/spreadsheets/d/1
                BxiMVs0XRA5nFMdKvBdBZjgmUUqptlbs74OgvE2upms/edit
91          spreadsheetId := "11RcJjyf5nNYkO7EbhurNYZl8JccohGouigOJkjNh5UA"
92          readRange := "D1"
93
94          writeLast := 0
95          resp, err := srv.Spreadsheets.Values.Get(spreadsheetId, readRange).Do()
96          if err != nil {
97                  log.Fatalf("Unable to retrieve data from sheet: %v", err)
98          }
99
100         if len(resp.Values) == 0 {
101                 fmt.Println("No data found.")
102                 return
103         } else {
104                 for _, row := range resp.Values {
105                         // Print columns A and E, which correspond to indices 0 and 4.
106                         strD1 := row[0].(string)
107                         //var dError
108                         writeLast, _ = strconv.Atoi(strD1)
109                         //fmt.Printf("%d\n", writeLast)
110                        // if(dError != nil){
111                        //         log.Fatalf("Unable to get data from D1: %v", dError)
112                        // }
113                 }
```

```
114         }
115
116         var vr sheets.ValueRange
117
118         rangeString := strconv.Itoa(writeLast)
119         writeRange := "A" + rangeString + ":C" + rangeString //+ ", D1"
120         myval := []interface{}{date, hour, state}
121         //vr.Values = append(vr.Values, myval)
122         oldSlice := vr.Values
123         vr.Values = append(vr.Values, myval)
124         _, err = srv.Spreadsheets.Values.Update(spreadsheetId, writeRange, &vr).
                ValueInputOption("RAW").Do()
125         if err != nil {
126                 log.Fatalf("Unable to write data to sheet. %v", err)
127         }
128
129
130         writeRange = "D1"
131         myval = []interface{}{strconv.Itoa(writeLast + 1)}
132         vr.Values = append(oldSlice, myval)
133         _, err = srv.Spreadsheets.Values.Update(spreadsheetId, writeRange, &vr).
                ValueInputOption("RAW").Do()
134         if err != nil {
135                 log.Fatalf("Unable to write data to sheet. %v", err)
136         }
137
138 }
```

Fonte: Autoria própria

5.1.2 O Aplicativo

Para construção do aplicativo em *Shiny-R*, de acordo com o Algoritmo 5 abaixo, primeiro é definido através da função de *dashboardPage* a cor do aplicativo, o *sidebar* e quais são as outras páginas, sendo estas criadas por *menuItem* que define as partes *Home*, Estatísticas e Quem somos.

Na função *dashboardBody* é definido quais são os dados que cada página irá receber, no caso isso é feito pela chamada de *tabItems*, como por exemplo, a data atual que está sendo atualizado, o *status* da fila, o gráfico com estatísticas dos últimos minutos, entre outras variáveis.

Logo em seguida é feito o desenvolvimento da página de "Quem somos", e para cada texto, é colocado uma caixa de texto através da função *box()*, que define o que será escrito, assim como sua cor e tamanho.

A partir daí é feita a requisição dos dados da tabela onde estão salvos os dados importantes para o projeto. É feito o *loadData* na função *funct()*, é enviado como parâmetro no campo *sheet key* o link da tabela do *Spreadsheet* que retorna a tabela toda para o *Shiny*, e no campo *names*(tabela) é dado os nomes para cada uma das colunas advindas do servidor, tendo "date"que representa a data em que foi feita a análise, e a "hora"e o

"estado"que é o *status* de cheio ou vazio.

Nomeando cada uma das colunas, é possível inferir as análises sobre os dados das mesmas, como por exemplo na página inicial, onde ocorrerá a busca da última data em que foi feito o monitoramento da fila, os horários para que seja possível inferir não somente a situação atual da fila, mas também a criação de um gráfico de movimento do R.U naquele dia, além de disponibilizar uma média de cheio ou vazio nos últimos cinco minutos verificados.

O gráfico é criado a partir da análise da coluna "estado"da tabela, onde todos os estados que são definidos como cheio terão os seus valores em 1, e os estados que forem vazios terão o valor definido como 0. Isso é feito através do *renderPlotly* em que é criada as variáveis para a montagem do gráfico em x e y, que é definido por *plotly* onde em x está a hora e y a coluna, em que como dito anteriormente os pontos irão variar entre 0 (vazio) e 1 (cheio).

Algoritmo 5- O aplicativo em R-*Shiny*

```
1
2  library(shiny)
3  library(shinydashboard)
4  library(googlesheets)
5  library(plotly)
6  library(stringr)
7
8  ui <- dashboardPage(skin = "red",
9      dashboardHeader(title = tags$img(src = 'russia.png', height = '65%')),
10     dashboardSidebar(collapsed = FALSE,
11         sidebarMenu(
12             menuItem("Home", tabName = "home", icon = icon("home")),
13             menuItem("Estatisticas", tabName = "estat", icon = icon("bar-chart")),
14             menuItem("Quem somos", tabName = "quem", icon = icon("address-card"))
15         )
16     ),
17     dashboardBody(
18         tabItems(
19             tabItem(tabName = "home",
20                 fluidRow(
21                     uiOutput("atualiza_data")
22                 )
23             ),
24             tabItem(tabName = "estat",
25                 plotlyOutput("grafico_estado"),
26                 br(),
27                 br(),
28                 fluidRow(
29
30                         valueBoxOutput("box_fila")
31
32                 )
33             ),
34             tabItem(tabName = "quem",
35                 box(
36                     title = "Projeto R.U.SSIA",
```

```
37                      status = "warning",
38                      collapsible = FALSE,
39                      h4(
40                          "Somos o Projeto R.U.SSIA, criado por amigos durante a disciplina
41                          de Projetos em Engenharia de Computacao na Universidade Federal
                                de Sao Paulo
42                          (UNIFESP)."
43                      )
44                  ),
45                  box(
46                      title = "Nossa missao",
47                      status = "warning",
48                      collapsible = FALSE,
49                      h4(
50                          "Com o objetivo de facilitar a vida dos estudantes de nossa
                                universidade, que chegam
51                          a passar horas nas filas do refeitorio universitario (RU),
                                criamos esse aplicativo."
52                      )
53                  ),
54                  box(
55                      title = "Menos filas",
56                      status = "warning",
57                      collapsible = FALSE,
58                      h4(
59                          "Hoje buscamos diminuir o tempo gasto pelos alunos que utilizam
                                da refeicao universitaria
60                          devido a necessidade de ficar tempo integral na faculdade esse
                                projeto foi desenvolvido, facilitando
61                          a transicao de informacoes entre a NUTRIMENTA e os alunos por
                                meio
62                          da disposicao dos dados sobre o quao cheio o RU se encontra."
63                      )
64                  ),
65                  box(
66                      title = "Equipe",
67                      status = "warning",
68                      collapsible = FALSE,
69                      h5("Bruno Ogata"),
70                      h5("Carolina Colla"),
71                      h5("Jaime Ossada"),
72                      h5("Joao Victor de Mesquita"),
73                      h5("Pedro Naresi"),
74                      h5("Raphael Faria")
75                  )
76              )
77          )
78      )
79  )
80
81  server <- function(input, output) {
82      tabela <- loadData()
83      names(tabela) <- c("date", "hora", "estado", "nil")
84      last_data <- nrow(tabela)
85
86      output$atualiza_data <- renderUI({
87          print(tabela$date[last_data])
88          box(title = paste("Fila do dia:", tabela$date[last_data]), status = "warning",
                collapsible = TRUE,
89              output$estado_fila <- renderValueBox({
```

```
90                  if(tabela$estado[last_data] == "cheio"){
91                      valueBox("Cheio", "O R.U. parece estar um pouco lotado no momento,
                            que tal ir daqui uns minutinhos?", icon = icon("thermometer-4"),
                            color = "red")
92                  } else {
93                      valueBox("Vazio", "Corre pro R.U.!", icon = icon("thumbs-up"), color
                            = "green")
94                  }
95              }),paste("Atualizado ultima vez as:", tabela$hora[last_data])
96
97          )
98      })
99
100     output$box_fila <- renderValueBox({
101         tab <- table(tabela$estado[(nrow(tabela)-10):nrow(tabela)])
102         count_cheio <- tab[names(tab) == "cheio"]
103         media <- count_cheio * 30
104
105         valueBox("Media de espera", paste(media, "segundos de fila cheia nos ultimos 5
                minutos"), icon = icon("hourglass-half"), color = "green")
106     })
107
108     output$grafico_estado <- renderPlotly({
109         estados <- tabela$estado
110         estado_v1 <- str_replace_all(estados, "cheio", "1")
111         estado_v1 <- str_replace_all(estado_v1, "vazio", "0")
112         estados_v2 <- as.numeric(estado_v1)
113         aux <- data.frame(hora = as.character(tabela$hora), estados = estados_v2)
114         plot_ly(aux, x = ~hora, y = ~estados, type = 'scatter', mode = 'lines')
115     })
116 }
117
118 loadData <- function(){
119     sheet_key <- "11RcJjyf5nNYkO7EbhurNYZl8JccohGouigOJkjNh5UA"
120     ss <- gs_key(sheet_key)
121     tabela <- gs_read(ss = ss)
122     return(as.data.frame(tabela))
123 }
124
125 # Run the application
126 shinyApp(ui = ui, server = server)
```

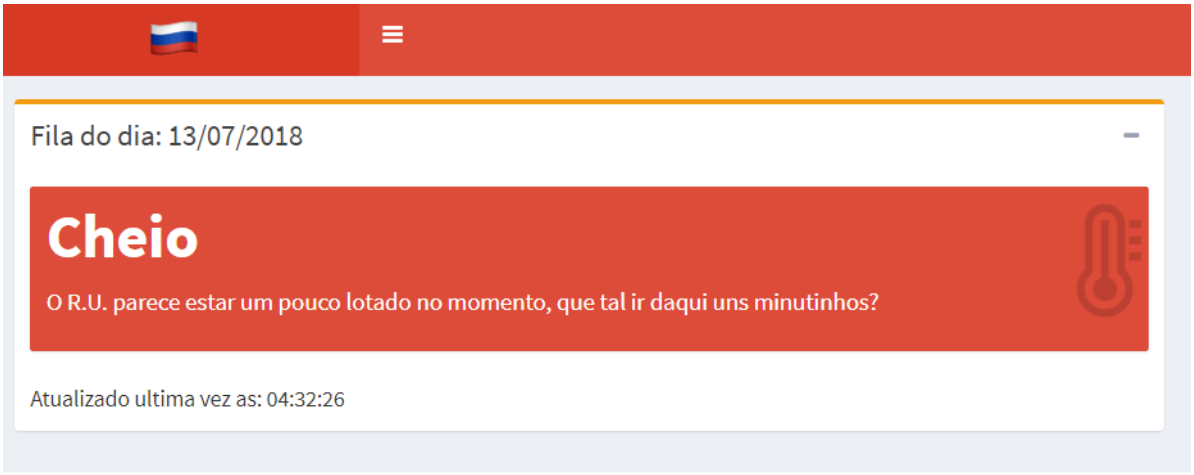
Fonte: Autoria própria

5.1.3 O Aplicativo Final

O Aplicativo final é composto por uma pagina inicial onde se encontra, como primeira informação, o estado da fila. Na Figura 10, por exemplo, a situação naquele momento seria considerado como a fila estando cheia. Outras informações que são passadas junto com o *status* da fila é o dia que está sendo avaliado e o momento que foi feito a ultima atualização de verificação do estado da fila.

Na parte superior da página, próximo ao nome do projeto, encontra-se o menu (Figura 11). Ao seleciona-lo seram mostradas as opções de estatísticas e quem somos. Estas

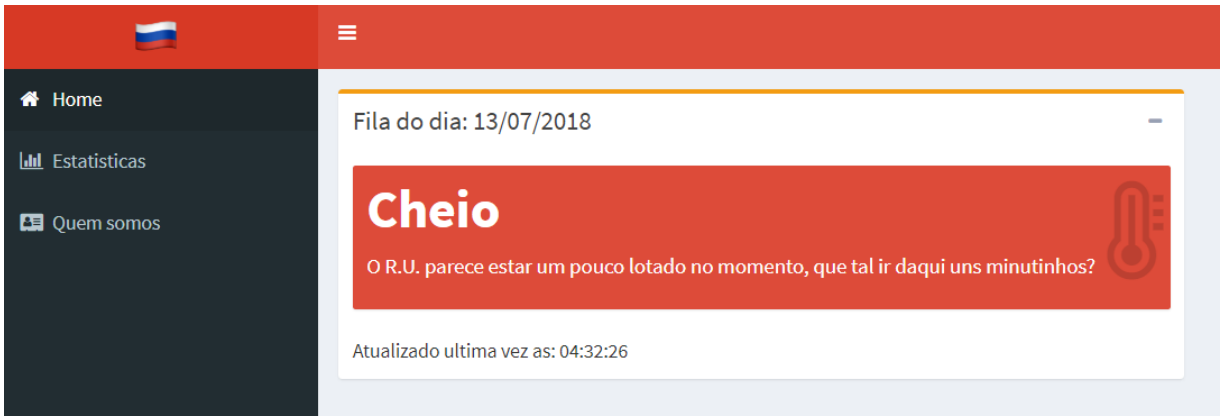
Figura 10 – Página principal do aplicativo



Fonte: Autoria própria

opções fazem um direcionamento para uma nova página que contém informações sobre o tópico selecionado.

Figura 11 – Menu do aplicativo



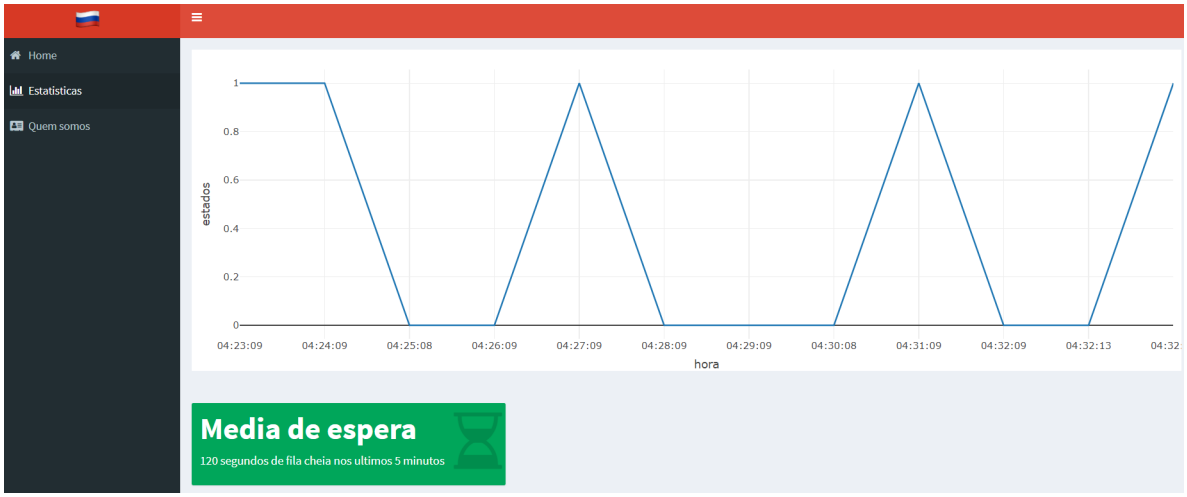
Fonte: Autoria própria

Quando selecionada a opção estatísticas será mostrado gráficos como o ilustrado na Figura 12. Esse gráfico mostra o estado do R.U. variado de acordo com cada horário. Assim quando o gráfico encontra-se com seu estado setado no valor 1 indica que neste horário o restaurante universitário se encontrava cheio, do contrário, quando setado como 0 o R.U. se encontra vazio.

Através destes dados também é possível inferir novas estatísticas com relação a média de espera, sendo que no caso do exemplo, a média de espera estava de 120 segundo durante os últimos 5 minutos avaliados.

A outra opção do menu é o quem somos, que serve para introduzir informações sobre o grupo que produziu o aplicativo, deixando também informações sobre a sua missão

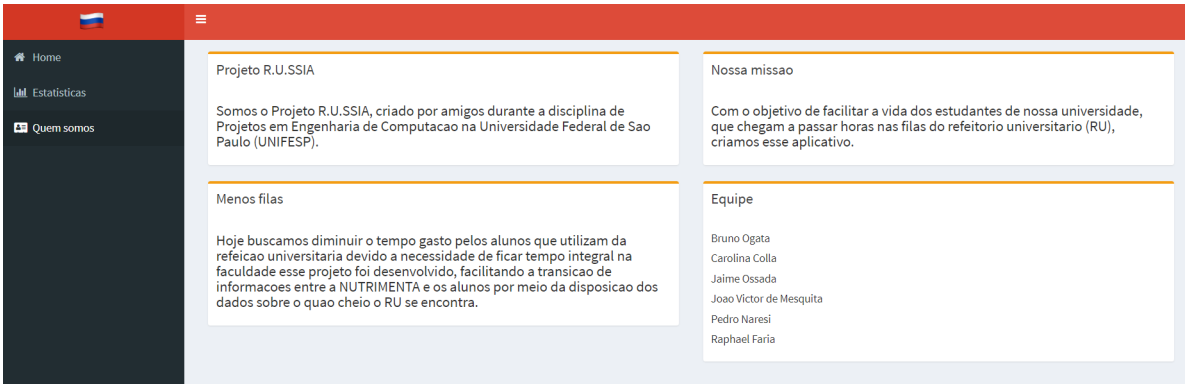
Figura 12 – Página de estatísticas do aplicativo



Fonte: Autoria própria

e objetivo com a criação do aplicativo (Figura 13).

Figura 13 – Página quem somos



Fonte: Autoria própria

6 Considerações Finais

O projeto de monitoramento de filas do restaurante universitário foi desenvolvido ao longo do semestre com o intuito de melhorara a rotina dos alunos da Unifesp, campos São José dos Campos, que possuem seu tempo de intervalo massivamente consumido pela necessidade de ficar em filas para poder realizar sua refeição.

Ao longo do semestre, uma estratégia foi traçada para alcançar esse objetivo e concluir com êxito o projeto, sendo ela a utilização de três câmeras para Arduíno posicionada entre as pilastras na extensão do corredor onde a fila se forma. Porém, essa estratégia logo se mostrou equivocada pois as câmeras adquiridas não possuem o circuito integrado FIFO (*First In First Out*), e necessitariam de um cartão de memória, acesso *Wi-Fi* independente e um Arduíno para cada câmera.

Esse fato acarretou uma mudança significativa do projeto que passou a adotar *Raspberry Pi* e uma câmera própria para esse componente como solução para essa dificuldade encontrada ao longo do processo. Essa solução possibilitou que o projeto pudesse ser finalizado dentro do prazo a pesar de o grupo ter tido um grande prejuízo com tempo devido a esse erro inicial.

O projeto como um todo funciona baseado em uma foto tirada pela câmera de *Raspberry Pi*, que envia essa imagem ao servidor de web, desenvolvido usando Go, que processa essa imagem calculando a distância euclidiana entre os seus *pixels* que, através da obtenção do erro médio, verifica a situação da fila e envia o resultado para uma tabela do *Google Spreadsheet*, sendo que essa tabela é usada pelo R onde é processada e virtualizada no *Shiny*. O resultado é mostrado por meio de um aplicativo que possui uma interface amigável para os usuários.

Em termos de futuras implementações, espera-se aplicar esse sistema para outros tipos de fila, fazendo com que ele seja versátil. Também espera-se adaptar o projeto para que seja possível visualizar quando as filas se encontram em um estado entre o cheio e o vazio, ou seja, um estado intermediário.

Referências

1 GARRETT, F. *Como funciona o Raspberry Pi? Entenda a tecnologia e sua aplicabilidade*. 2014. Disponível em: <http://www.techtudo.com.br/noticias/noticia/2014/11/como-funciona-o-raspberry-pi-entenda-tecnologia-e-sua-aplicabilidade.html>. Acesso em: 13/07/2018. Citado na página 11.

2 R, C. de. *Curso R*. 2018. Disponível em: <http://material.curso-r.com/shiny/>. Acesso em: 13/07/2018. Citado na página 12.

3 RYCHLEWSKI, F. *Trabalhando com Go (GoLang), a linguagem do Google*. 2016. Disponível em: <https://imasters.com.br/back-end/trabalhando-com-go-golang-a-linguagem-do-google>. Acesso em: 13/07/2018. Citado na página 13.

4 NET, C. *O que é um servidor em computação?* 2016. Disponível em: <https://www.controle.net/faq/o-que-sao-servidores>. Acesso em: 13/07/2018. Citado na página 14.

5 VIEIRA, N. *Entendendo um pouco mais sobre o protocolo HTTP*. 2007. Disponível em: <https://nandovieira.com.br/entendendo-um-pouco-mais-sobre-o-protocolo-http>. Acesso em: 10/07/2018. Citado na página 18.

6 EIS, D. *CSS3 – Breve introdução ao RGBA*. 2011. Disponível em: <https://tableless.com.br/css3-breve-introducao-a-rgba/>. Acesso em: 12/07/2018. Citado na página 20.

ANEXO A – Anexo 1

A.1 Jaime Ossada

Eu e mais dois membros do grupo ficamos responsáveis pelo Servidor Go e a comunicação com o Raspberry Pi. Foram feitos um servidor na Linguagem Go, que se comunica com o Spreadsheets do Google com a API fornecida pela própria Google. Também foi feito um script Go que foi rodado periodicamente no Raspberry Pi para se enviar uma foto para o servidor, sendo necessária a configuração do Raspberry para se fazer essa comunicação com o servidor.

Uma das maiores surpresas durante o desenvolvimento foi a facilidade de se criar um servidor Local com Go, bastando algumas simples linhas de códigos, sem a necessidade de instalação de algo externo, como o Apache. Uma dificuldade encontrada durante grande parte do projeto foi a tentativa de fazer esse envio de foto pelo Arduino, que foi em seguida trocado pelo Raspberry, que facilitou muito na captura da foto, sendo possível utilizar simplesmente linguagens de mais alto níveis para se tirar a foto.

Outra dificuldade foi utilizar a API do Spreadsheets, que para Go especificamente é muito mal-documentada com exemplos incompletos, tendo sido gasto bastante tempo para entender como se utilizar a API em si, mas nada que tenha impedido a conclusão do Projeto.

O Desenvolvimento dessa parte do projeto foi fluida, tendo reuniões semanais com os outros responsáveis por essa parte constantemente, mantendo um contato próximo já que o desenvolvimento dessa parte foi feita completamente em conjunto, diferente do que foi a parte de aplicativo.

A.2 Carolina Locatelli Colla

Minha experiência com o projeto foi positivo. Inicialmente o grupo passo por grandes dificuldades e a mudança de estratégia, ocasionada no meio do semestre, abalou os integrantes que enfrentaram problemas com o tempo ao longo do restante do projeto. Porém, como essa dificuldade retirou o grupo de sua zona de conforto, a mudança acabou agregando extremo valor tanto ao projeto quanto ao desenvolvimento pessoal de cada integrante que teve de aprender a lidar com contratempos, novas técnicas e linguagens que tiveram de ser aplicadas sobre o projeto.

Fiquei responsável pela parte de servidor junto com o Raphael Ribeiro e o Jaime Ossada. O código do servidor exigiu que nos reuníssemos sempre que possível pelo fato

de, diferente do restante do grupo que desenvolveu o aplicativo, o código não poder ser feito de maneira independente. Assim, foi necessário uma boa programação dos alunos que tiveram que se programar para encontrar horários em comum para desenvolver o projeto de forma adequada.

Com relação a experiência acredito que as principais conclusões que pude tirar foi com relação ao componente utilizado no projeto, no caso o *Raspberry Pi*, que facilitou o projeto de forma significativa. O projeto com Arduíno exigia a montagem complexa e um computador que deveria estar constantemente conectado ao mecanismo. Isso iria contra o principio da disciplina que visa criar algo aplicável a realidade.

Outra observação obtida foi a implementação do servidor local com Go. Essa linguagem facilitou o projeto e simplificou a implementação desejada. Foi através dele que foi feito o tratamento de imagem que utilizava a cor de pixel e a distancia euclidiana para determinar o erro médio e assim determinar o estado da fila no momento em que a foto foi tirada.

O que considero como maior dificuldade no processo de desenvolvimento do projeto foi a utilização do *Google Spreadsheet*. O problema encontrado foi advindo do fato de sua documentação ser escassa, sendo encontrado pouca informação, com pouca explicação e na maioria das vezes imcompleta.

Acredito que o projeto me ajudou a aprender a lidar com imprevisto e a importância de reuniões de grupo para um bom desenvolvimento do trabalho.

A.3 Raphael Ribeiro Faria

A proposta da disciplina de criar um projeto na área de computação com os conhecimentos adquiridos ao longo da universidade até agora se mostrou muito interessante, incentivando os alunos a colocarem suas ideias em prática através de projetos. Inicialmente, a primeira proposta do grupo era de desenvolver uma aplicação voltada para a área de produção de cervejas, projetando um sistema capaz de controlar algumas etapas da fabricação.

Como a ideia acabou se apresentando de difícil execução, o integrante Bruno Ogata teve a ideia de desenvolver uma aplicação que monitore a situação atual da fila do R.U., verificando e mostrando se a mesma, naquele momento, se encontra cheia ou vazia. A primeira tentativa era de se fazer o trabalho utilizando o microcontrolador Arduino e módulos de câmera, ideia que não foi levada para frente devido ao problema de as câmeras não possuírem o módulo específico para o processamento de imagens, dificultando muito o prosseguimento do projeto.

A alternativa foi utilizar o microcomputador *Raspberry pi*, que com o auxílio de

uma câmera específica, envia as fotos tiradas para um servidor local que em seguida, envia os dados correspondentes para uma tabela para que o aplicativo possa consultar a situação atual do restaurante.

Eu particularmente fiquei responsável pela implementação do servidor, que realiza esse processamento das imagens para determinar se há pessoas ou não na fila, trabalhando em conjunto com os demais integrantes Jaime Ossada e Carolina Locatelli, onde foram realizadas reuniões para que os códigos necessários fossem implementados em conjunto. A linguagem escolhida para o desenvolvimento foi a linguagem *Go*, devido à sua facilidade de criação de um servidor a partir de seus recursos.

Foi preciso criar arquivos responsáveis por realizar a requisição HTTP, para representar o servidor em si, para realizar o processamento das imagens e para escrever os dados na tabela desejada. Uma dificuldade encontrada pelo grupo foi a sincronização do servidor com a tabela utilizada no *spreadsheet*, devido à dificuldade encontrada com a documentação do mesmo, e a minha dificuldade pessoal foi em relação ao processamento das imagens, visto que ela calcula o erro médio entre os *pixels* e verifica o seu valor para determinar a situação atual do restaurante, etapa essa que demandou uma certa atenção para ser consolidada.

A.4 João Victor de Mesquita

A experiência com o projeto desde o início se demonstrou ser um desafio, partindo do problema que atinge a todos que frequentam o Restaurante Universitário (R.U) o desenvolvimento do projeto não foi só para alguém ou para uma disciplina e sim para todos do grupo. Desde o início tivemos o problema com as câmeras do arduíno e a dificuldade do Arduíno em tirar fotos e processar as mesmas, após a resolução do problema utilizando *Raspberry Pi* tivemos outro empecilho que era o aplicativo, desde início trabalhei com o foco no aplicativo mesmo, de início nós como grupo tivemos a ideia de utilizar o *AppyPie* o que não se tornou viável. Dei a ideia de utilizarmos o *framework Ionic*, entretanto junto com a ajuda do Bruno Ogata vimos que também era uma ideia a ser descartada.

Partindo disso juntamente com o Pedro Naresi e o Bruno Ogata, focamos na parte de criação do aplicativo utilizando linguagem R onde seria possível a construção do aplicativo em *Shiny*. O Bruno nos guiou pois é o membro que mais tem experiência utilizando essas ferramentas, porém não foi algo difícil de aprender e implementar, trabalhamos em conjunto a distância, onde criamos de início a base bruta do aplicativo e só após de tudo pronto foi feito a conexão com a tabela do *SpreadSheet*.

A conexão com a tabela do *SpreadSheet* foi feita apenas por último pois era preciso com que de fato o servidor armazenasse os dados na tabela pois a partir daí teríamos a certeza de que a lógica que está por trás do funcionamento do projeto está funcionando.

A experiência com o *R-Shiny* foi muito boa pois com isso foi possível aprofundar o conhecimento na ferramenta e também seu uso com tabelas externas. No geral trabalhar em grupo e trabalhar com um projeto utilizando a computação em favor das outras pessoas foi muito positiva e que agregou muito conhecimento tanto em planejamento de um trabalho, comunicação e maior contato com ferramentas e conteúdos que já vimos em outras unidades curriculares e muito além.

Acredito que para os próximos passos seria a real implementação do projeto com o foco de trazer uma melhoria para a comunidade do ICT que frequenta diariamente o restaurante e fazendo assim com que o trabalho desenvolvido nesta UC de Projetos em Engenharia de Computação avance muito mais do que ficar restrito em uma sala de aula.

A.5 Bruno Ogata

Quando o professor Tiago de Oliveira propôs o trabalho, logo vi que seria um projeto desafiador e complexo, porém gratificante. Durante a fase da definição do tema muitas ideias surgiram e todas me deixavam animado e logo queria que fosse dado o início em seu desenvolvimento. Uma destas ideias foi o Sistema de Monitoramento de uma Cervejeira que, infelizmente, logo foi descartada devido a locomoção até Campos do Jordão (cidade o qual a Cervejeira fica localizada) ser difícil e poderia acarretar adversidades futuras. A ideia do Monitoramento da Fila veio do estresse enfrentado na maioria dos dias e a noção de que nosso projeto poderia facilitar a vida da maioria dos ICeTianos foram fatores decisivos para escolha do tema. Além disso, poucas são as disciplinas que tem cunho prático e que a criatividade do aluno é a peça fundamental para que a aprovação aconteça, características essas que acho muito importante em minha vida acadêmica.
As adversidades encontradas durante a fase de execução me ajudaram a perceber como eu lido com dificuldades inesperadas e percebo que a criticidade para a procura de uma nova solução fez com que eu e até mesmo todos do grupo (arrisco dizer) amadurecemos como profissionais e que provavelmente estes mesmos problemas não voltaram a ocorrer novamente em nossos trabalhos. A adversidade mais caricata que enfrentamos foi a compra das três câmeras do Arduino sem FIFO. Vimos que se tivéssemos tido uma melhor fase de planejamento e um sólido plano de execução não teríamos enfrentado este problema e não teríamos desperdiçado alguns reais de investimento e tempo com a procura de tutoriais para como fazê-los funcionar em uma máquina.
Ficar responsável pelo desenvolvimento do aplicativo não foi um problema. Já trabalho com R e Shiny há um tempo e sabia que estas tecnologias tinham muito a agregar em nosso projeto. Junto de Pedro e João Victor, implementamos o aplicativo buscando sempre eficiência e que o usuário tivesse uma ótima experiência em suas visitas, metas que acreditamos que foram atingidas. Uma de minhas preocupações era o acesso ao banco de dados *Spreadsheet*, API antes que nunca havia utilizado e, traumatizado por uma

falha tentativa de acesso ao banco de dados *Firebase* (também da Google) em projetos passados, acreditei que dificuldades seriam enfrentadas nesta etapa. Para me preocupar mais, durante as discussões com o grupo, o Jaime comentou que a documentação da API do Google Spreadsheet para Go era muito ruim e imaginei que para o R seria o mesmo. Para minha surpresa (e sorte) a documentação do pacote "googlesheet" para o R é muito boa e a conexão com o banco de dados foi feito sem grandes problemas.
Ao final deste projeto, considero que muito me foi agregado profissionalmente, pois a vivência de um trabalho de Engenheiro de Computação em um projeto real foi algo que não tive em nenhuma outra disciplina. Além disso, trabalhar com amigos é sempre algo especial, o que fez do R.U.SSIA ser um projeto também muito especial em um semestre que foi muito caótico. Agradecimentos especiais a Vitor Hugo Barbedo que emprestou a câmera do Raspberry Pi e nos deu muitas dicas para mexer na tecnologia, salvando o R.U.SSIA da falência múltipla dos órgãos.

A.6 Pedro Naresi

No momento de definição do tema escolhido para o desenvolvimento do projeto várias ideias vieram a mente do grupo, desde monitoramento de fabricação de cervejas ate analisadores de desmatamentos. Após definirmos que nosso projeto seria um sistema de monitoramento das cervejeiras fomos coletar informações e sobre o funcionamento e como poderíamos realizar os testes. Após receber as informações vindas de um parente de um membro do grupo (Jaime) descobrimos que o projeto da cervejeira inteligente seria de difícil implementação devido a não portabilidade do sistema de produção e também pelo custo dos sensores. Por fim, nos direcionamos para outras ideias, foi ai que surgiu o projeto R.U.SSIA que possui o objeto de monitorar e verificar a fila do nossa famigerado "RU" (restaurante universitário). Para o desenvolvimento do projeto R.U.SSIA nos dividimos em dois grupos, um responsável por fazer toda a programação do servidor (*backend*) e uma responsável pela parte do aplicativo que sera disponibilizado para o usuário. Eu, Bruno Ogata e João ficamos responsáveis por essa parte. O plano de ação foi o desenvolvimento da interface a partir do *framework* Shiny. O mesmo funciona com a linguagem programação R e nos permite o desenvolvimento de gráficos e janelas intuitivas e que promovem a facilidade ao interagir com o usuário final. Ao receber os dados do servidor (que criar uma tabela csv no google spreadsheets) verificamos a integridade dos mesmos e os "plotamos" no website disponível para as aplicações Shiny. Assim, todos os usuários que possuírem o link terão acesso direto através de qualquer plataforma (celular, tablet, computador). O desenvolvimento do projeto se mostrou tranquilo e com fluidez. Todos os participantes do grupo sabem trabalhar em grupo e entendem as limitações de cada um, nos proporcionando uma experiencia de desenvolvimento muito fluida e sem "stress".