



**MINISTÉRIO DA EDUCAÇÃO
UNIVERSIDADE FEDERAL DE GOIÁS
ESCOLA DE ENGENHARIA ELÉTRICA, MECÂNICA E DE
COMPUTAÇÃO**



GABRIEL AUGUSTO DE VITO D'ABBADIA GUIIMARÃES

**RELATÓRIO DO ESTÁGIO SUPERVISIONADO REALIZADO NA
POLICHAT SOLUÇÕES E WEB LTDA**

Goiânia

2018



MINISTÉRIO DA EDUCAÇÃO
UNIVERSIDADE FEDERAL DE GOIÁS
ESCOLA DE ENGENHARIA ELÉTRICA, MECÂNICA E DE
COMPUTAÇÃO



GABRIEL AUGUSTO DE VITO D'ABBADIA GUIMARÃES

RELATÓRIO DO ESTÁGIO SUPERVISIONADO REALIZADO NA POLICHAT SOLUÇÕES E WEB LTDA

Relatório do Estágio Supervisionado apresentado à Escola de Engenharia Elétrica, Mecânica e de Computação da Universidade Federal de Goiás, como requisito parcial para a integralização do curso em Engenharia de Computação.

Orientador: Prof. Sandrerley Ramos Pires

Supervisor: Alberto da Silva Filho

Goiânia

2018

Lista de Figuras

Figura 1 Exemplo de rotas.....	9
Figura 2 Exemplo de URL da Polichat, seus breadcrumbs, ícone associado e nome traduzido	10
Figura 3 Código que verifica a URL, a transforma em um array e começa a verificação	11
Figura 4 Verifica a posição do breadcrumb.....	11
Figura 5 Função que associa a URL a um título em português e a associa a um ícone	12
Figura 6 Relatório sem filtros por canais e por departamento.....	13
Figura 7 Uma das funções de filtro por canal.....	14
Figura 8 Função de filtro por departamento	14
Figura 9 Parte da função do controller que realiza o filtro por departamento e por canal	15
Figura 10 Alguns commits do bitbucket que estão na branch 'Reports'	15
Figura 11 Relatório com os canais clicáveis (funcionam como filtros)	16
Figura 12 Gráfico em pizza respeitando a tabela de filtro por departamento.....	16
Figura 13 Gráfico de histórico de atendimentos.....	16
Figura 14 Relatórios por atendente.....	17
Figura 15 Exemplo de resposta de uma requisição do tipo GET à API	19
Figura 16 Interface gráfica desenvolvida para criar novos clients e novos tokens de acesso ..	20
Figura 17 Commits do bitbucket	20
Figura 18 Algoritmo para agrupar o chat_history e trazer o número de interações corretamente	21
Figura 19 Informações adicionadas ao plano	22
Figura 20 Código responsável por realizar as consultas necessárias no banco de dados	23
Figura 21 Número de usuários online.....	23
Figura 22 Consulta da quantidade de usuários em fila.....	23
Figura 23 Opção de canal do tipo 'whatsapp'	25
Figura 24 Opção de canal do tipo e-mail.....	25
Figura 25 Opção de canal do tipo e-mail.....	26
Figura 26 Colunas do banco de dados da tabela channel_configs	27
Figura 27 Colunas do banco de dados da tabela chat_history	27
Figura 28 Função responsável por testar o SMTP.....	28
Figura 29 Função responsável por gerenciar o recebimento de um e-mail	28
Figura 30 Função responsável por gerenciar a rotina de CRON	29
Figura 31 Função que dita o que deve ser feito ao receber um e-mail	29
Figura 32 Commits do Bitbucket.....	30
Figura 33 Roadmap da Polichat no Trello.....	31

Sumário

Introdução	5
1 Breadcrumbs automáticos	8
1.1 Definição.....	8
1.2 Problema	8
1.3 Solução.....	10
2 Melhorias nos relatórios (24/08/2018 – 03/09/2018)	13
2.1 Definição.....	13
2.2 Problema	13
2.3 Solução.....	13
2.4 Resultados	15
3 Criação de uma API de Integração	18
3.1 Definição.....	18
3.2 Problema	18
3.3 Solução.....	18
3.4 Resultados	19
4 Alterações no algoritmo do número de interações	21
4.1 Definição.....	21
4.2 Problema	21
4.3 Solução.....	21
5 Integração com o E-Mail	24
5.1 Definição.....	24
5.2 Problema	24
5.3 Solução.....	24
5.3.1 Frontend.....	24
5.3.2 Banco de dados	26
5.3.3 Controllers	27
5.3.4 Alterações no Chat.....	29
Conclusão	30
Referências	32

Introdução

A Polichat é uma startup fundada em março de 2018 e possui um software de mesmo nome da empresa. O software é um omnichannel com o objetivo de melhorar o atendimento prestado por empresas via principais redes sociais atuais (Whatsapp, Facebook, Instagram, E-Mail).

Através da plataforma, a empresa consegue, por exemplo, transformar um número de Whatsapp em uma central de atendimento ao cliente, contabilizando os chamados, encerrando-os, transferindo-os para outros atendentes, com direito a histórico de chamados por atendente, relatórios dos atendimentos (gráficos contendo o número de chamados encerrados, em andamento, finalizados por encaminhamento e na triagem), estatísticas (tempo médio de primeira resposta, pior tempo de primeira resposta, tempo médio para concluir um chamado, avaliações do atendente), chatbot para encaminhar o cliente para o setor desejado, disparos de mensagens em massa.

O software foi criado a partir da necessidade, dentro de uma clínica hospitalar, de gerenciar seus atendimentos realizados via Whatsapp. Nesta clínica, muitos pacientes entravam em contato com a empresa e apenas uma pessoa não conseguia atender a todos os clientes. Foi necessário, então, criar uma plataforma que ajudasse nisso, transformando um número de Whatsapp em um canal de atendimentos.

O software pode ser acessado através do link <https://polichat.com.br>.



Figura 1 Landing Page do site da Polichat

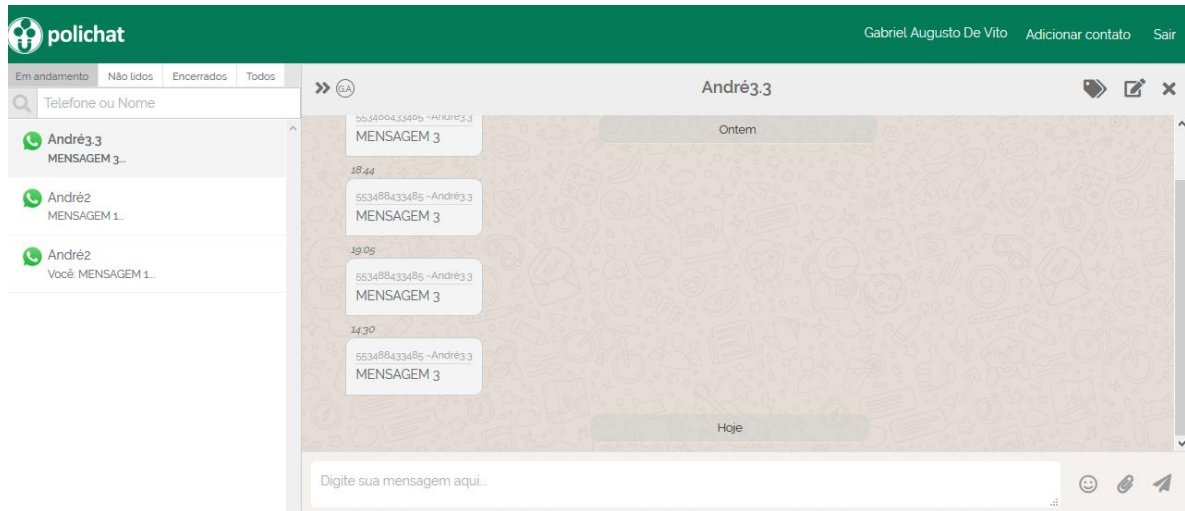


Figura 2 Página do Atendente do chat



Figura 3 Canais cadastrados pela empresa

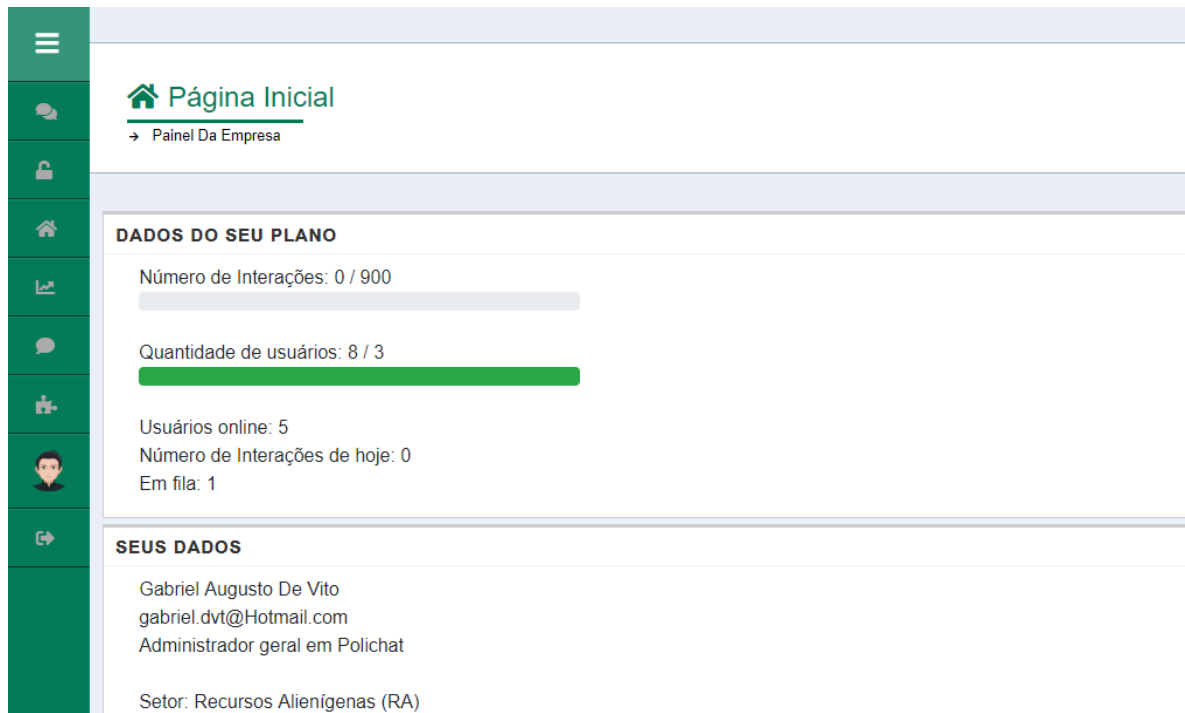


Figura 4 Página inicial da empresa

1 Breadcrumbs automáticos

1.1 Definição

Breadcrumbs, **breadcrumb navigation** ou **breadcrumb trail** (literalmente, navegação por migalhas de pão ou trilha de migalhas de pão, em alusão à história de João e Maria) é um sistema de navegação estrutural usada para proporcionar ao usuário um meio de localização dentro da estrutura de navegação do site.

Breadcrumbs indicam a localização atual do usuário e quais são os níveis superiores da estrutura. Normalmente em uma barra horizontal na parte superior do site, permitem rastrear o caminho de volta para uma seção maior.

A **navegação por breadcrumbs** melhora a usabilidade do site, pois diminui o esforço do usuário para atingir uma página e facilita sua navegação, ao refletir a ordem lógica do conteúdo.

1.2 Problema

No sistema da Polichat, os breadcrumbs existiam, porém eram adicionados manualmente, página a página do sistema. Em outras palavras, sempre que nós criávamos uma nova view, deveríamos adicionar este breadcrumb na página, sendo que isso poderia ser automatizado, uma vez que as URLs de todo o sistema segue o **padrão RESTful**.

A imagem mostra a interface de usuário do sistema Polichat. No topo, há uma barra de navegação com o ícone de um usuário e o texto 'Usuários'. Abaixo disso, uma barra de breadcrumbs indica o caminho: 'Painel Da Empresa / Usuários / 163'. O formulário principal, intitulado 'POLICHAT - CADASTRAR USUÁRIO', contém os seguintes campos: 'Voltar para o painel da empresa' (link azul), 'Cargo' (menu suspenso com 'Operador de chat' selecionado), 'Setores' (com opções 'Recursos Alienígenas (RA)', 'Contabilidade' e 'setor teste' desmarcadas), 'Nome' (campo de texto) e 'Email' (campo de texto).

Figura 5 Exemplo de Breadcrumb dentro do Sistema de Polichat

O sistema é cheio de rotas seguindo o padrão restful, como pode ser visto em um exemplo de código abaixo:


```

/*****
Gerenciamento
*****/
Route::group(['prefix' => 'users'], function(){
    Route::get('/{customer}', 'UsersController@index');
    Route::get('/create/{customer}', 'UsersController@create');
    Route::post('/create/{customer}', 'UsersController@store');
    Route::post('/update/{customer}', 'UsersController@update');
    Route::get('/{customer}/{user}', 'UsersController@show');
    Route::get('/{customer}/{user}/edit', 'UsersController@edit');
    Route::post('/{customer}/{user}/change_status', 'UsersController@change_status');
});

```

Figura 1 Exemplo de rotas

No exemplo acima, percebemos que existem rotas do tipo:

- /users/1
- /users/create/1
- /users/1/2
- /users/1/2/edit
- /users/1/2/change_status

Nosso algoritmo deve ser capaz de criar breadcrumbs para cada uma dessas rotas.

Portanto, se o usuário estiver em /users/1/2/edit, deve existir um breadcrumb que leva para /users, outro que leva para /users/1, e outro que leve para /users/1/2.

Outras observações importantes:

- A rota raiz deve estar em todos os links.
- Os nomes das rotas estão em inglês e devem ser traduzidos para o português, pois o título da página deve estar em português.
- Cada nome de rota deve estar relacionada a um icon do font-awesome (<https://fontawesome.com/>), pois todas as páginas possuem um ícone.

A figura abaixo ilustra este exemplo.

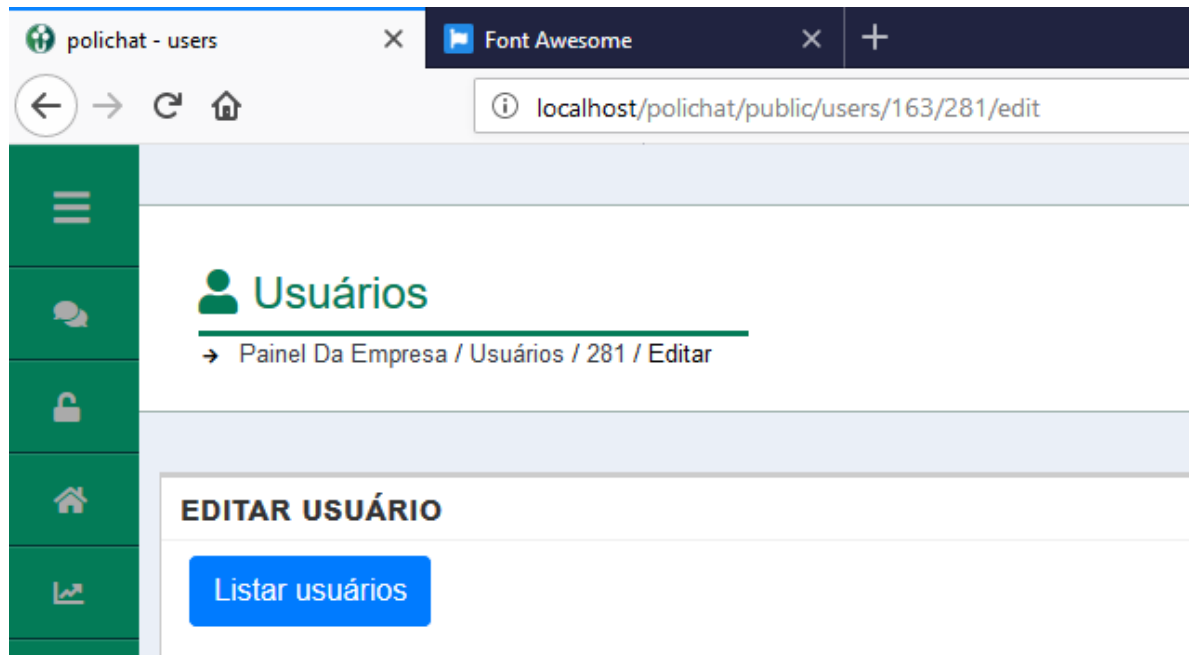


Figura 2 Exemplo de URL da Polichat, seus breadcrumbs, ícone associado e nome traduzido

Perceba a presença do ícone do font-awesome, da tradução do nome 'users' para 'Usuários' e da presença dos breadcrumbs anteriores quando se entra na url desejada.

A ideia aqui é que nunca tenhamos que programar essas breadcrumbs, e que elas sejam geradas de forma automática conforme a URL.

1.3 Solução

Para isso, fiz um script em PHP que fizesse a leitura da URL atual (que sempre segue o padrão RETful, caso contrário seria muito difícil fazer um script que atendesse a todas as rotas do sistema), separasse cada nome separado por '/' em um array e verificasse cada valor deste array para ir montando um outro array (que contém os nomes das breadcrumbs, a URL e seu ícone).

Este script foi inserido no primeiro arquivo lido pelo sistema, que é pai de todas as outras views, para forçar com que todas as URLs executem-lo.

Os scripts abaixo foram os scripts desenvolvidos nesses primeiros dias de estágio, capazes de realizar este trabalho.

```

/*
 *   Verifica a URL e, a partir dela, preenche uma breadcrumb
 */
// Particiona a URL
$breadcrumb = str_replace(url("/")."/", "", Request::url());
$breadcrumb = explode('/', $breadcrumb);

// Definição do título
if ($breadcrumb[0] == 'admin' && !empty($breadcrumb[1])) {
    $title = $breadcrumb[1];
}
else {
    $title = $breadcrumb[0];
}

```

Figura 3 Código que verifica a URL, a transforma em um array e começa a verificação

```

103 // Percorre todos os níveis da URL
104 foreach($breadcrumb as $j => $b) {
105     // Primeiro termo da URL
106     if ($j == 0)
107     {
108         // Home + Alguma outra coisa
109         if ($breadcrumb[$j] == 'admin') {
110             $url[0] = $breadcrumb[0];
111             $name[$j] = 'Administração';
112             $url[1] = $breadcrumb[1] ?? null; // Pouco importa esta url, não é utilizada
113             $name[1] = $breadcrumb[1] ?? null;
114         }
115         elseif ($breadcrumb[$j] != 'home') {
116             $url[$j] = 'customers/'. (isset($breadcrumb[1]) ? $breadcrumb[1] : Auth::user()->customer_id);
117             $name[$j] = 'Painel da Empresa';
118             $url[1] = $breadcrumb[0]; // Pouco importa esta url, não é utilizada
119             $name[1] = $breadcrumb[0];
120         }
121         // Apenas home
122     } else {
123         $url[$j] = 'home';
124         $name[$j] = 'Painel da Empresa';
125     }
126 }
127
128 // Segundo termo da URL
129 if($j == 1)
130 {
131     $url[$j] = $breadcrumb[0] . '/' . $breadcrumb[1];
132     $name[$j] = $breadcrumb[0];
133
134     // Ex: /users/create/163 => /users/163
135     /*
136     * if (empty($breadcrumb[2]) && !is_nan($breadcrumb[2]) && empty($breadcrumb[3]))
137     * {
138     *     $url[$j] = $breadcrumb[0] . '/' . $breadcrumb[2];
139     * }
140     */
141 }
142
143 if ($j == 2)
144 {
145     $url[$j] = $breadcrumb[0] . '/' . $breadcrumb[1] . '/' . $breadcrumb[2];
146     $name[$j] = $breadcrumb[2];
147 }
148
149 if ($j == 3)
150 {
151     $url[$j] = $url; // Não é url
152     $name[$j] = $breadcrumb[3];
153 }
154 }
155

```

Figura 4 Verifica a posição do breadcrumb

Para realizar a tradução do nome em inglês para o título, criei a função abaixo, que é utilizada conforme a próxima figura.

```
function translate_route_app_breadcrumbs($name, $icon = false) {
    /* Coloque aqui o nome da url, o título desejado e o ícone da página seguindo o font-awesome */
    $names = [
        'admin'          => ['Administração Polichat', 'fas fa-unlock fa-text'],
        'home'           => ['Página Inicial', 'fas fa-home fa-text'],
        'contacts'        => ['Todos os Contatos', 'fas fa-users fa-text'],
        'report'          => ['Relatórios', 'fas fa-chart-line fa-text'],
        'edit'            => ['Editar', 'fas fa-home fa-text'],
        'settings'        => ['Configurações Globais', 'fas fa-cogs fa-text'],
        'customers'        => ['Clientes', 'fas fa-user fa-text'],
        'channels'        => ['Canais', 'fas fa-comments fa-text'],
        'users'           => ['Usuários', 'fas fa-user fa-text'],
        'dados_empresa'  => ['Dados da Empresa', 'fas fa-building fa-text'],
        'dados_plano'     => ['Dados do plano', 'fas fa-cubes fa-text'],
        'dados_pagamento' => ['Dados do pagamento', 'fas fa-credit-card fa-text'],
        'roles'           => ['Cargos', 'fab fa-cloudsmith fa-text'],
        'payments'        => ['Pagamentos', 'fas fa-credit-card fa-text'],
        'create'          => ['Novo', 'fab fa-cloudsmith fa-text'],
        'permissions'     => ['Permissões', 'fas fa-clipboard-list fa-text'],
        'campaigns'       => ['Campanhas', 'fa fa-unlock fa-text'],
        'departments'     => ['Departamentos', 'fa fa-unlock fa-text'],
        'card'            => ['Cobrança no Cartão', 'fa fa-credit-card fa-text'],
        'messages'        => ['Mensagens', 'fa fa-comment fa-text'],
        'integrations'     => ['Integrações', 'fas fa-puzzle-piece'],
        'emails'          => ['Configurações do E-mail', 'fa fa-envelope']
    ];

    // translate an icon
    if ($icon) {
        if (array_key_exists($name, $names)) {
            return $names[$name][1];
        } else {
            return 'home';
        }
    }

    // translate a name
    } else {
        if (array_key_exists($name, $names)) {
            return $names[$name][0];
        } else {
            return $name;
        }
    }
}
```

Figura 5 Função que associa a URL a um título em português e a associa a um ícone

2 Melhorias nos relatórios (24/08/2018 – 03/09/2018)

2.1 Definição

Meu segundo trabalho como desenvolvedor dentro da Polichat foi fazer uma melhoria em seus relatórios.

Para explicar um pouco melhor quais foram as melhorias realizadas no sistema de relatórios, é importante mostrar quais são as partes que constituem o relatório.

O relatório possui um gráfico em pizza trazendo estatísticas sobre os atendimentos realizados pela empresa (quantidade de chamados não atendidos, encerrados, em atendimento, em fila e total de chats). Também conta com um gráfico em linhas, mostrando esses mesmos dados por período, em uma linha do tempo.

Também conta com um relatório de atendimentos por usuário.

2.2 Problema

Através de um campo de data inicial e data final, o usuário tinha a opção de filtrar esses dados por período. Este, até então, era o único tipo de filtro que o usuário poderia utilizar. Houve, então, a necessidade de adicionar um filtro **por canal** e um outro filtro por **usuários de um determinado departamento**.

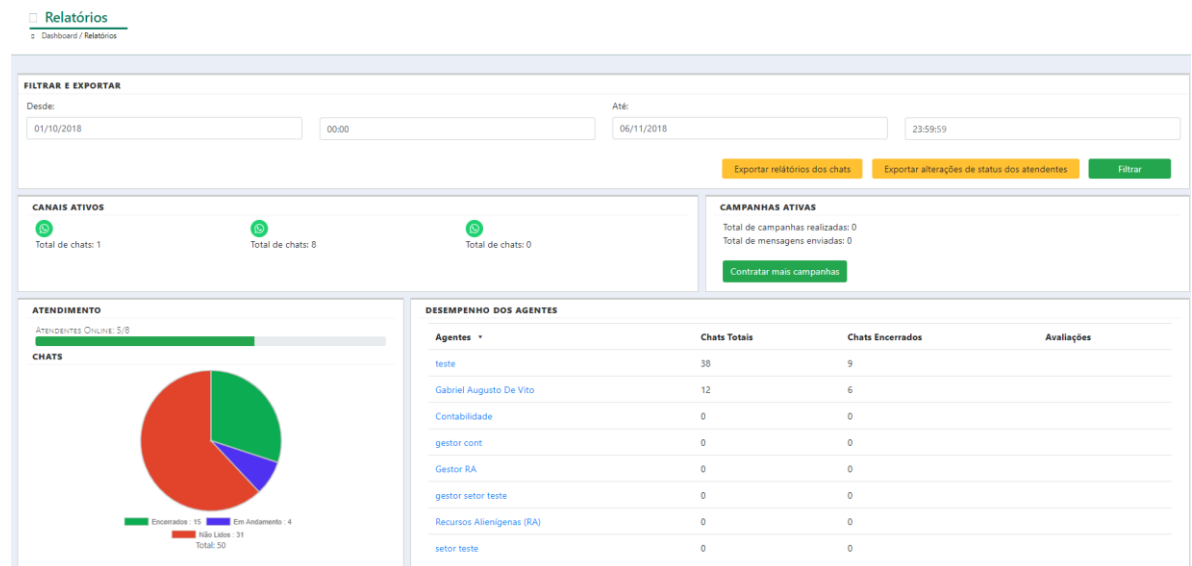


Figura 6 Relatório sem filtros por canais e por departamento

2.3 Solução

Adicionar esses dois filtros não é tão simples, pois envolve alterar o controller (ReportsController), a URL da página, que contém a consulta via GET com parâmetros (ids dos canais a serem buscados, filtro por data e ids dos usuários pertencentes aos departamentos filtrados), alterar o frontend, verificar quais filtros estão ativos e quais não estão ativos, e trabalhar no arquivos com extensão .js responsável por montar os gráficos.

```

/**
 * Filtra o chat_history por canal
 */
public function filterByChannel($channel_id, $chat_history)
{
    //filtro por canal
    if (is_null($channel_id) || $channel_id == "") {
        $channel_id = [];
        $chat_history = $chat_history;
    } else {
        $channel_id = explode(',', $channel_id);
        $chat_history = (clone $chat_history)->whereIn('channel_id', $channel_id);
    }

    $arr = [$channel_id, $chat_history];
    return $arr;
}

```

Figura 7 Uma das funções de filtro por canal

```

/** Filtra o chat_history por departamento */
public function filterByDepartment($chat_history, $ids_users_from_department, $department_id)
{
    if (!(is_null($department_id) || $department_id == "")) {
        //busca todos os chats de todos os usuários deste departamento, inclusive os que possuem origin_id nulo
        $chat_history = ((clone $chat_history)->whereRaw("(origin_id is null or origin_id in (" . implode(',', $ids_users_from_department) .
    })

    return $chat_history;
}

```

Figura 8 Função de filtro por departamento

O primeiro passo foi buscar, no banco de dados, quais os canais existentes para este cliente (utilizando o Eloquent ORM do Laravel, realizando uma consulta simples no banco de dados). Com posse disso, apresentar esses canais no frontend, utilizando HTML (Blade Template do Laravel) e CSS (Bootstrap). Essa foi a parte simples.

Para buscar o total de chats de cada canal, foi necessária uma outra consulta, que não vem ao caso agora.

Cada canal possui um checkbox e foi adicionado um novo botão de ‘filtrar’. Vale lembrar que toda a página de relatórios, incluindo todos os filtros, é apenas um formulário, que será disparado pelo botão ‘filtrar’ via GET.

Ao disparar, a requisição pega todos os filtros, monta a URL e faz uma requisição na mesma página. A rota irá acionar o método responsável do controller, que irá recarregar a página trazendo as consultas necessárias.

Ao recarregar a página, com os parâmetros conforme a consulta, o arquivo javascript se encarrega de montar os gráficos com os dados trazidos do backend.

Para montar a tabela de departamentos, tive que buscar, primeiramente, os departamentos. Em seguida, trazer os agentes (usuários que estão dentro de cada departamento) conforme o filtro ‘departments’, realizado pelo input do tipo select do HTML.

Deve-se atentar que tanto o gráfico de pizza quanto o gráfico de linhas agora possuem esses dois filtros a mais, e tiveram que ser tratados para trazerem os dados corretos.

Ambos esses gráficos são preenchidos com consultas de algumas tabelas: a primeira (e principal), é a tabela ‘chat_history’, que armazena todas as interações entre usuários do sistema e seus clientes, via chat. A consulta, que antes recebia apenas o filtro de data, agora passa a receber esses outros dois filtros.

A figura abaixo mostra parte do ‘ReportsController’, mostrando o tipo de consulta realizado com os filtros por canal e por departamento.

```
//busca todos os canais
$channels = channel_customer::where('customer_id', '=', $customer->id)->get();

//busca todos os departamentos deste customer (deste cliente)
$departments = Department::where('customer_id', $customer->id)->get();

//pesquisa o chat_history
$chat_history = chat_history::where([
    ['customer_id', '=', $customer->id],
    ['created_at', '>=', $date_start],
    ['created_at', '<=', $date_end],
]);

$ch = $chat_history->get();

//apenas para trazer os nomes dos canais
$chat_history2 = (clone $ch);

//filtra o chat_history por canal
$arr = $this->filterByChannel($channel_id, $chat_history);
$channel_id = $arr[0];
$chat_history = $arr[1];

//ids dos usuários do departamento selecionado
$ids_users_from_department = $this->getUsersByDepartment($chat_history, $department_id, $customer);

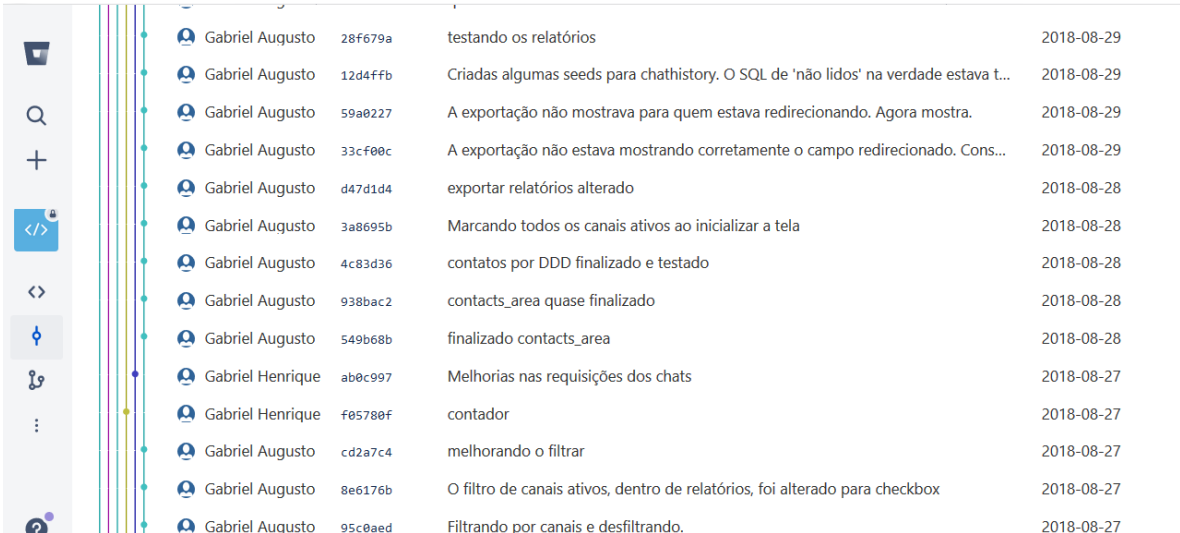
//filtra por departamento
$chat_history = $this->filterByDepartment($chat_history, $ids_users_from_department, $department_id);

//número de interações
$number_of_interactions = $this->getNumberOfInteractions((clone $chat_history)->get());
```

Figura 9 Parte da função do controller que realiza o filtro por departamento e por canal

2.4 Resultados

Alguns commits do Bitbucket são mostrados na figura abaixo.



Gabriel Augusto	28f679a	testando os relatórios	2018-08-29
Gabriel Augusto	12d4ffb	Criadas algumas seeds para chathistory. O SQL de 'não lidos' na verdade estava t...	2018-08-29
Gabriel Augusto	59a0227	A exportação não mostrava para quem estava redirecionando. Agora mostra.	2018-08-29
Gabriel Augusto	33cf00c	A exportação não estava mostrando corretamente o campo redirecionado. Cons...	2018-08-29
Gabriel Augusto	d47d1d4	exportar relatórios alterado	2018-08-28
Gabriel Augusto	3a8695b	Marcando todos os canais ativos ao inicializar a tela	2018-08-28
Gabriel Augusto	4c83d36	contatos por DDD finalizado e testado	2018-08-28
Gabriel Augusto	938bac2	contacts_area quase finalizado	2018-08-28
Gabriel Augusto	549b68b	finalizado contacts_area	2018-08-28
Gabriel Henrique	ab0c997	Melhorias nas requisições dos chats	2018-08-27
Gabriel Henrique	f05780f	contador	2018-08-27
Gabriel Augusto	cd2a7c4	melhorando o filtrar	2018-08-27
Gabriel Augusto	8e6176b	O filtro de canais ativos, dentro de relatórios, foi alterado para checkbox	2018-08-27
Gabriel Augusto	95c0aed	Filtrando por canais e desfiltrando.	2018-08-27

Figura 10 Alguns commits do bitbucket que estão na branch ‘Reports’

O resultado final da página de relatórios ficou da seguinte maneira:

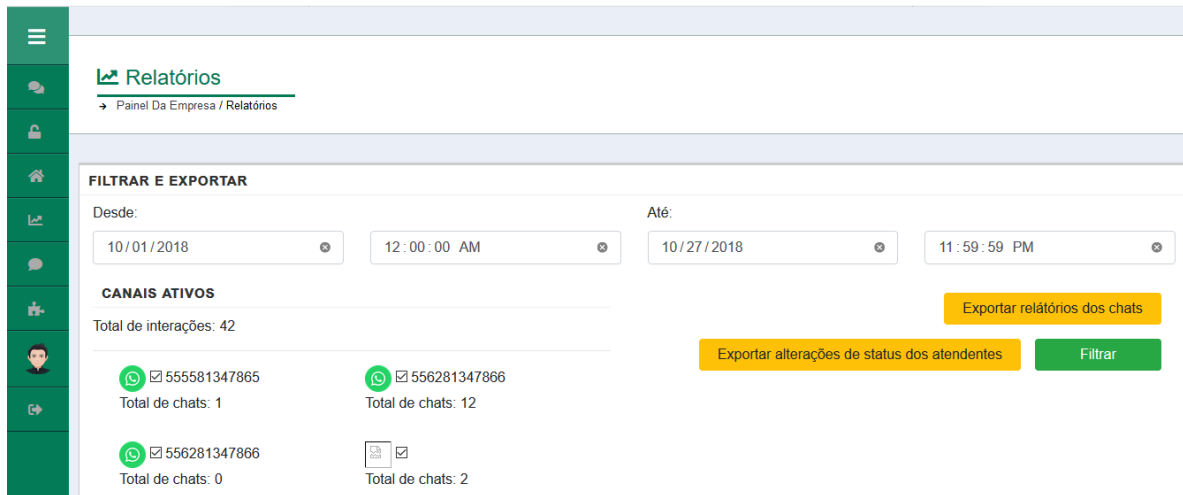


Figura 11 Relatório com os canais clicáveis (funcionam como filtros)



Figura 12 Gráfico em pizza respeitando a tabela de filtro por departamento.

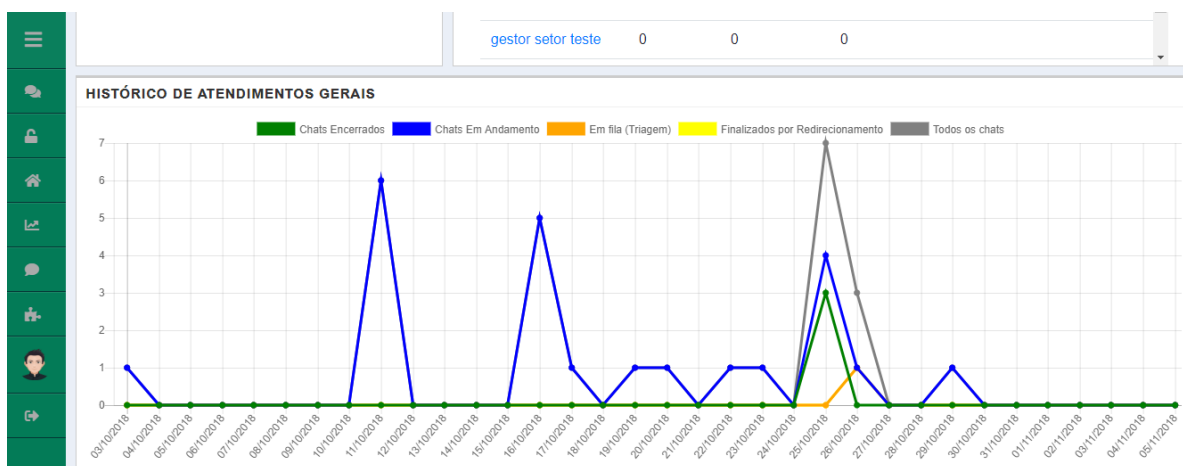


Figura 13 Gráfico de histórico de atendimentos



Figura 14 Relatórios por atendente

3 Criação de uma API de Integração

3.1 Definição

Grande parte das empresas hoje estão criando suas próprias APIs (Application Programming Interface) para permitirem que as funcionalidades de seus softwares sejam integradas por outros sistemas e, assim, fazerem parcerias com outras empresas e permitirem com que as funcionalidades principais de um software se complementem com as funcionalidades de outros softwares.

3.2 Problema

Até o momento, a Polichat não contava com nenhuma API de integração. Desta forma, perdia muitos clientes que necessitavam fazer integrações com o sistema ou parcerias. Hoje em dia, dificilmente um sistema robusto não libera uma API para que outros sistemas integrem.

3.3 Solução

Na Polichat, desenvolvi uma API de Integração, que permite com que outros sistemas utilizam as suas funcionalidades. Esta API utiliza a tecnologia OAuth 2.0 para autenticação, prezando pela segurança das informações passadas entre os softwares.

A documentação completa da API foi gerada pelo software Postman (<https://www.getpostman.com/>) e está publicada no nosso site (<https://polichat.com.br/developer>).

A nossa API é capaz de:

- **mostrar, criar, editar e excluir** canais da empresa;
- criar um novo chat;
- fazer com que uma mensagem apareça como lida no chat;
- mostrar os chats abertos;
- mostrar os chats por usuário;
- mostrar as mensagens de um chat;
- mostrar a galeria de um chat;
- **mostrar, criar, editar e excluir** um contato;
- redirecionar a conversa de um contato para outro operador;
- abrir um chat com um contato;
- fechar um chat com um contato;
- enviar mensagens para um contato;

Todas as requisições à API são feitas através de uma URL, respeitando os parâmetros e o tipo de requisição (GET, POST, DELETE, UPDATE).

Como utilizamos um Framework MVC (Laravel) para desenvolver o sistema e a API, essa requisição vai para uma rota, que irá dizer qual método de qual controller será executado para trazer os dados buscados.

Conforme escrito na documentação, todas as URLs do tipo GET da API podem utilizar até três parâmetros: **filters** (seleciona colunas específicas da consulta), **sort** (realiza uma ordenação por ordem crescente ou decrescente) ou **algum filtro específico** (passa-se o nome da coluna e o valor para buscar um registro específico).

Exemplos de requisições GET com parâmetros:

<http://www.polichat.com.br/api/v1/customers/{id}?name=teste>
<http://www.polichat.com.br/api/v1/customers/{id}?name=teste&sort=-id>
http://www.polichat.com.br/api/v1/customers/{id}/chats?filters=id,customer_id,category_id,teste

3.4 Resultados

Os resultados desta API de integração ainda estão por vir. Alguns clientes já fecharam conosco devido à integração a estão testando.

Exemplo de resposta à requisição para mostrar todos os canais:

```
{
  "current_page": 1,
  "data": [
    {
      "id": 177,
      "channel_id": 1,
      "customer_id": 163,
      "phone": "556281347866",
      "url": null,
      "name": null,
      "type": "whatsapp",
      "status": null,
      "history": null,
      "created_at": "2018-08-21 17:11:06",
      "updated_at": "2018-08-23 17:45:19"
    },
    {
      "id": 179,
      "channel_id": 1,
      "customer_id": 163,
      "phone": "556281000000",
      "url": null,
      "name": "teste",
      "type": "whatsapp",
      "status": {
        "id": 20736,
        "customer_id": 163,
        "channel_id": 179,
        "qr_code": null,
        "type": 0,
        "status": null,
        "uid": "556281000000@c.us",
        "history": 20,
        "created_at": "2018-08-27 14:08:11",

```

Figura 15 Exemplo de resposta de uma requisição do tipo GET à API

Para que o usuário utilize esta API, é necessária uma autenticação via token. O Laravel Framework utiliza o “Passport”, que se autentica com OAuth2.0.

Para gerenciar esses tokens (gerar, verificar ou excluir um token de autenticação), desenvolvi a tela mostrada na figura abaixo.

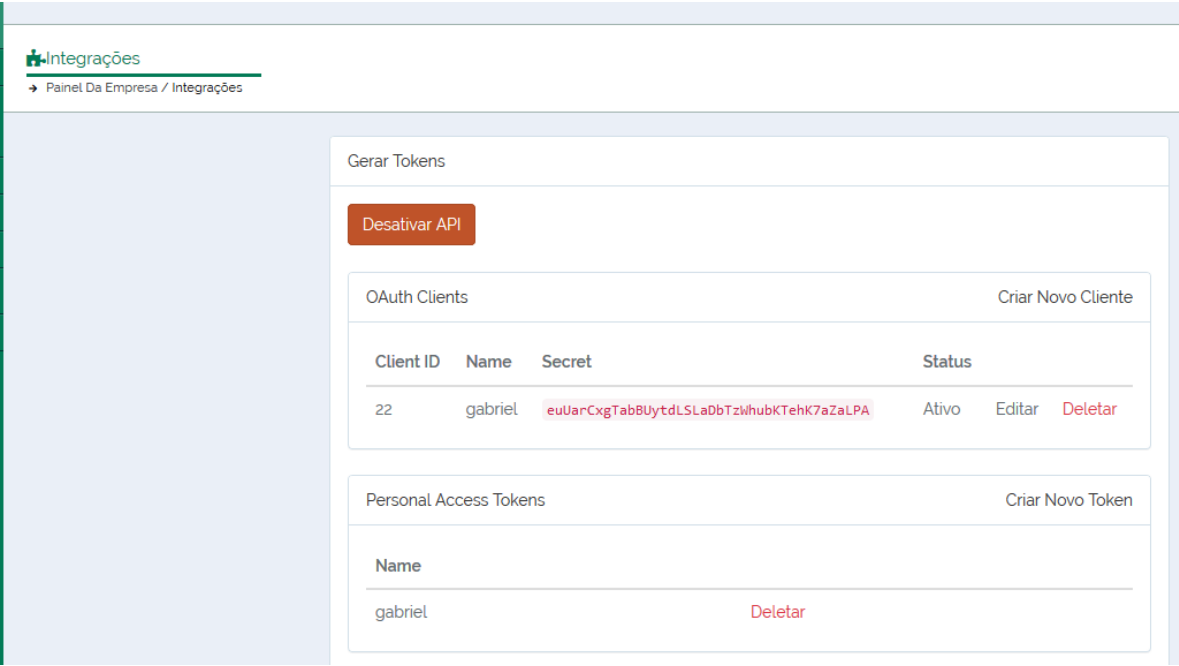


Figura 16 Interface gráfica desenvolvida para criar novos clients e novos tokens de acesso

Alguns commits do bitbucket são mostrados na figura abaixo.

Author	Commit	Message	Date
Gabriel Augusto	33cf08c	A exportação não estava mostrando corretamente o campo redirecionado. Consertado.	2018-08-29
Gabriel Augusto	047d104	exportar relatórios alterado	2018-08-28
Gabriel Augusto	3a86950	Marcando todos os canais ativos ao inicializar a tela	2018-08-28
Gabriel Augusto	4c83d36	contatos por DDD finalizado e testado	2018-08-28
Gabriel Augusto	938bac2	contacts_area quase finalizado	2018-08-28
Gabriel Augusto	549e68b	finalizado contacts_area	2018-08-28
Gabriel Henrique	a08c997	Melhorias nas requisições dos chats	2018-08-27
Gabriel Henrique	f95780f	contador	2018-08-27
Gabriel Augusto	cd2a7c4	melhorando o filtrar	2018-08-27
Gabriel Augusto	8e6176b	O filtro de canais ativos, dentro de relatórios, foi alterado para checkbox	2018-08-27
Gabriel Augusto	95c0aed	Filtrando por canais e desfiltrando.	2018-08-27
Gabriel Augusto	13c5585	adicionado o filtro por canal.	2018-08-27
Gabriel Henrique	a10a00a	Merge branch 'master' of https://bitbucket.org/polichat/polichat-web-app	2018-08-27
Gabriel Henrique	a031bf6	correção na listagem de contatos	2018-08-27
Gabriel Henrique	86cc9c0	Merge branch 'develop' of https://bitbucket.org/polichat/polichat-web-app into develop	2018-08-26
Gabriel Henrique	8efa5c4	Merge branch 'master' of https://bitbucket.org/polichat/polichat-web-app	2018-08-26
Gabriel Henrique	cc85090	.	2018-08-26
Gabriel Augusto	4a82f9d	Em relatórios, filtrando por setor	2018-08-24
Gabriel Henrique	f6c90b0	Merge branch 'master' of https://bitbucket.org/polichat/polichat-web-app	2018-08-23
Gabriel Henrique	5695ae7	Correções no contador de chats por usuário	2018-08-23
Gabriel Augusto	f547027	iniciando as alterações nos relatórios: colocar o número do whatsapp nos canais ativos	2018-08-23
Gabriel Augusto	ee40b53	finalizando os breadcumbs.	2018-08-23
Gabriel Augusto	901fd19	breadcrumb de campanhas consertada	2018-08-23

Figura 17 Commits do bitbucket

4 Alterações no algoritmo do número de interações

4.1 Definição

Um conceito bastante importante neste sistema é o de **interações**.

Interações são definidas, via contrato com o cliente, como uma conversa entre um contato e um atendente no intervalo de 24 horas. Sendo assim, caso um chamado não seja finalizado dentro de um dia, assim que o contato falar novamente com o atendente após às 0h deste dia, é contada uma nova interação. Caso o contato seja transferido de um atendente para outro, não é contabilizada uma segunda interação. Porém, caso o atendente encerre o chamado e o contato entre novamente em contato, é contabilizada uma nova interação.

Interações são fundamentais, pois a Polichat realiza a cobrança pela sua quantidade e também pela quantidade de gestores e operadores de chat.

4.2 Problema

Havia um equívoco no número de interações até o momento em que eu entrei na Polichat. O número de interações não estava sendo limitado ao dia. Sendo assim, a Polichat contabilizava como apenas uma interação mesmo quando o usuário entrava em contato novamente no outro dia, o que causava um certo prejuízo para a empresa, uma vez que ela deixava de ganhar por interações.

4.3 Solução

Foi necessário realizar um agrupamento por contato e por data na tabela ‘chat_history’, limitando em 24 horas. Isso foi feito utilizando métodos da Collection do Laravel (<https://laravel.com/docs/5.7/collections#available-methods>), junto ao PHP, da seguinte maneira:

```
/*
 * Buscando o número total de interações
 */
$chat_history = chat_history::where([
    ['customer_id', '=', $customer->id],
    ['created_at', '>=', $date_start],
    ['created_at', '<=', $date_end],
]);

$ch = $chat_history->get();
//agrupa o histórico de chats por contato
$chat_history_grouped = $ch->groupBy('contact_id');
$numberOfInteractions = 0;
$contacts_dates = [];

//percorre cada chat_history agrupado por contato
foreach($chat_history_grouped as $ch_contact) {
    //caso tenha mais de um chat aberto para este contato, devemos verificar as suas datas e não contabilizar as interações no mesmo dia
    if (sizeof($ch_contact) > 1) { //possui mais de uma interação no mesmo dia
        //estou agora agrupando por dia. Fazendo isso para contabilizar apenas uma vez os chats que possuem o mesmo dia
        $ch_contact = $ch_contact->groupBy(function($obj) {
            return Carbon::parse($obj->created_at)->format('Y-m-d');
        });
        //após agrupado, a cada grupo devemos contabilizar apenas uma vez.
        $numberOfInteractions += sizeof($ch_contact);
    } else { //caso tenha apenas um chat aberto para este contato
        $numberOfInteractions++;
    }
}
```

Figura 18 Algoritmo para agrupar o chat_history e trazer o número de interações corretamente

Desta forma, o algoritmo se fez correto e o problema de número de interações foi resolvido.

Outro recurso desenvolvido neste mesmo período foi adicionar, na página inicial do gestor, uma barra de progresso com o número de interações até o momento e o número de interações máxima definida pelo plano contratado.

Junto a esta barra de progresso, foi adicionada a quantidade de usuários que esta empresa possui, relacionando com a quantidade máxima de usuários que este cliente tem direito.

Também a quantidade de usuários online no momento, o número de interações no dia e o número de usuários em fila no dia. Os usuários em fila são aqueles que aguardam atendimento, portanto é um número bastante importante para o gestor, pois se este número for muito elevado, ele pode tomar algumas atitudes em relação à sua equipe de atendimento.

A figura abaixo mostra os campos adicionados.

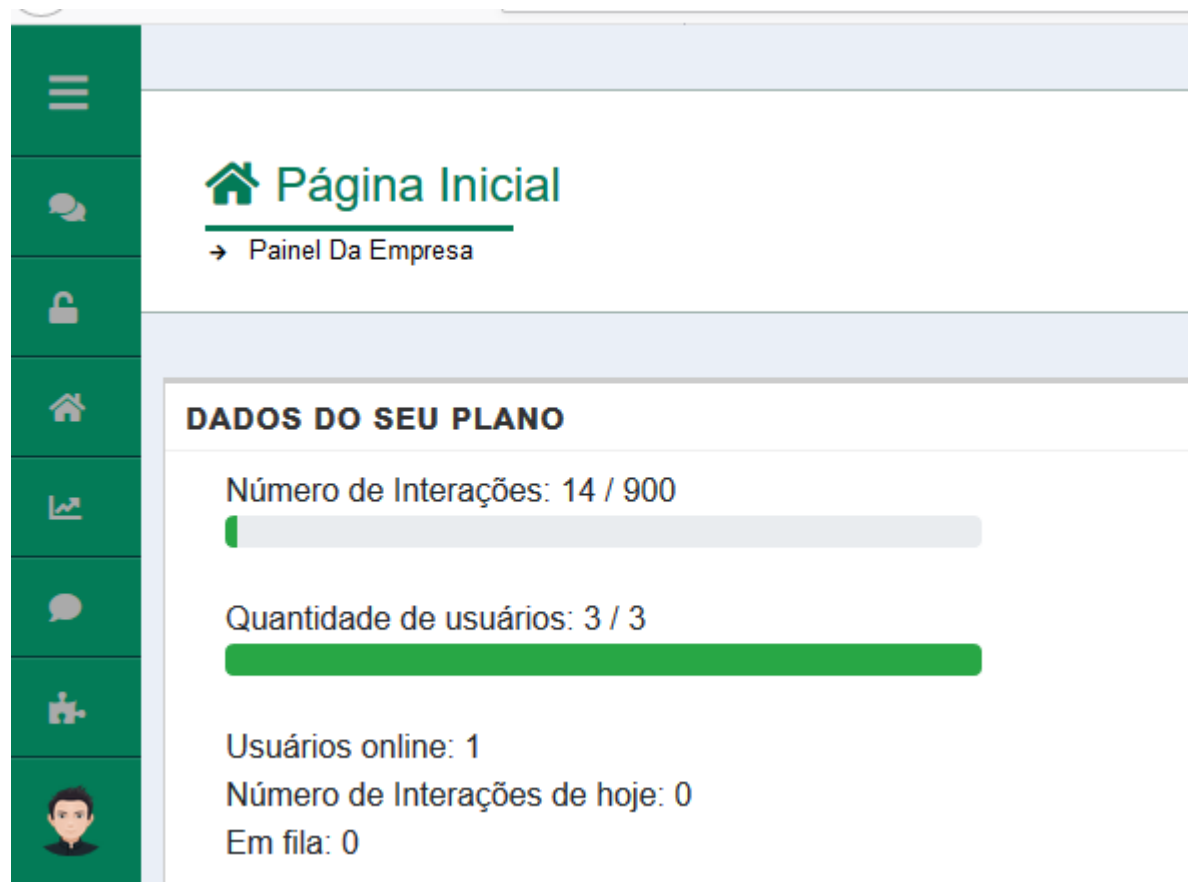


Figura 19 Informações adicionadas ao plano

Para buscar o número máximo de interações do plano do cliente, utilizamos buscas no banco de dados de forma simples, relacionando a tabela 'service_plain' com o id do plano do cliente. A imagem abaixo mostra como a consulta é realizada.

```

$maxInteractions = 0;
$maxAgent = 0;
$maxAdmin = 0;
$numberOfUsers = 0;

if($customer->plain) {
    $services_plain = service_plain::where('plain_id',$customer->plain->id)->get();
    foreach($services_plain as &$service_plain) {
        $service_plain->service_name = $service_plain->service->name;
    }
    $agents_service = $services_plain->where('service_name','agents_simult')->first();
    $managers_service = $services_plain->where('service_name','agents_manager')->first();
    $whatsapp = $services_plain->where('service_name','whatsapp')->first();

    $customer->max_agents = $agents_service->interactions + $managers_service->interactions;
    $customer->max_channels = $whatsapp->channels;

    $maxInteractions = $services_plain->where('service_id', 1)->last()->interactions ?? null;
    $maxAgent = $services_plain->where('service_id', 8)->last()->interactions ?? null;
    $maxAdmin = $services_plain->where('service_id', 9)->last()->interactions ?? null;
    $numberOfUsers = User::where('customer_id', Auth::user()->customer_id)->count();
}else{
    $customer->max_agents = 100;
    $customer->max_channels = 100;
}

```

Figura 20 Código responsável por realizar as consultas necessárias no banco de dados

Nesta mesma consulta são buscados o número máximo de agentes no plano e o número máximo de administradores no plano. Com posse desses dois números, podemos definir o número máximo de usuários.

O número de usuários online é, também, uma simples consulta no banco de dados utilizando a cláusula 'where', relacionando o id do cliente com o status 1.

```

// busca o número de usuários online
$numberOfOnlineUsers = User::where([
    ['customer_id', '=', $customer->id],
    ['status', '=', 1]
])->count();

```

Figura 21 Número de usuários online

Para mostrar os dados do número de usuários em fila, basta realizar uma consulta em 'chat_history', da seguinte maneira:

```

/** Busca quantos estão em fila */
public function inRow($customer)
{
    $inRow = chat_history::where([
        ['customer_id', '=', $customer->id],
        ['origin_id', '=', null]
    ])->count();

    return $inRow;
}

```

Figura 22 Consulta da quantidade de usuários em fila

5 Integração com o E-Mail

5.1 Definição

A Polichat, como dito antes, tem o objetivo não apenas de ser uma plataforma web capaz de fazer atendimentos via Whatsapp, mas sim se tornar um omnichannel completo, com integração ao E-mail, Facebook, Instagram e outras possíveis redes sociais futuras.

Por este motivo, fez-se necessário a integração com o E-mail, de tal forma que o e-mail possa ser utilizado como um novo canal e, desta forma, permitir ao usuário a enviar e receber e-mails através da mesma tela de chat do Whatsapp.

A ideia é que o usuário não precise acessar o site do seu provedor de e-mails e nem saia da plataforma para receber, enviar ou responder aos e-mails.

5.2 Problema

Até o momento do desenvolvimento da integração com o e-mail, o único tipo de canal possível era o Whatsapp. Desta forma, o usuário não era capaz de enviar ou receber e-mails através da plataforma.

5.3 Solução


Desenvolver esta integração foi, até então, a tarefa que mais demorei implementar na Polichat. Isso porque existiam conceitos que até então eu não estava familiarizado.

Conceitos como: **SMTP, IMAP, POP3**. Estes conceitos foram sendo adquiridos ao longo do desenvolvimento da integração. Faz-se necessário o uso do SMTP para enviar e-mails e IMAP para receber os e-mails.

5.3.1 Frontend

A primeira alteração foi no cadastro de novos canais, permitindo o e-mail como um novo tipo.

Para isso, foi necessário fazer alterações no frontend, adicionando um novo tipo no 'select' e criando uma regra para que, quando o usuário escolhesse 'email' como opção, trouxesse à tona os campos de usuário, e-mail e senha do e-mail, e configurações do servidor, como endereço IMAP, SMTP, Porta de cada um deles e tipo de certificado (SSL ou TLS). A figura abaixo mostra o resultado final da parte visual.



→ Painel Da Empresa / Canais / Novo

CRIAR CANAL

Listar canais

Rede social do canal

whatsapp

Nome do canal

Ex.: Canal para suporte financeiro

Telefone com DDD e código de país


Ex.: 5562980000000

Quantidade de mensagens para backup

7

Confirmar

Figura 23 Opção de canal do tipo 'whatsapp'



→ Painel Da Empresa / Canais / Novo

CRIAR CANAL

Listar canais

Rede social do canal

email

Nome do canal

Ex.: Canal para suporte financeiro

Buscar os e-mails de quanto tempo atrás? (dias)

7

Informações da Conta de E-Mail

Usuário

Figura 24 Opção de canal do tipo e-mail



Informações da Conta de E-Mail

Usuário

E-mail

Senha

Informações do Servidor

IMAP ? Porta IMAP ? Certificado ?

SMTP ? Porta SMTP ? Certificado ?

Testar Conexão Confirmar

Figura 25 Opção de canal do tipo e-mail

O botão de ‘Testar conexão’ faz uma requisição no servidor IMAP e SMTP, com as suas respectivas configurações, e é capaz de dizer se as configurações estão válidas ou não.

Isso é feito graças ao uso da biblioteca do PHP denominada PHPMailer. Esta biblioteca é de domínio público e está publicada no Github.

Junto a esta biblioteca, existem algumas classes desenvolvidas internamente pela empresa, capaz de facilitar as configurações de e-mail.

Uma dessas classes é a IncomingEmails, que possui métodos de testar a conexão, inicializar uma nova conexão com o servidor de e-mails, buscar os e-mails (isso pode ser configurado, para buscar os e-mails lidos ou não lidos), percorrer esses e-mails e transferí-los para uma outra classe desenvolvida por mim, capaz de trabalhar nesses dados vindos do e-mail para que possamos integrar com o nosso sistema.

5.3.2 Banco de dados

No banco de dados, foi necessário adicionar uma tabela denominada ‘channel_configs’, capaz de armazenar as configurações do canal cadastradas no formulário da imagem acima (SMTP, IMAP, E-mail, Senha, Certificados, usuário, ID do canal, etc). A imagem abaixo mostra as colunas criadas:

Colunas:										
#	Nome	Tipo de dados	Tamanho/It...	Unsign...	Permitir...	Zero fill	Padrão	Comentário	Colaço	Expressão
1	id	INT	10	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	AUTO_INCREMENT			
2	channel_id	INT	10	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	Nenhum padrão			
3	username	VARCHAR	255	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	Nenhum padrão		utf8mb4_unicode_ci	
4	email	VARCHAR	50	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	Nenhum padrão		utf8mb4_unicode_ci	
5	password	VARCHAR	100	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	Nenhum padrão		utf8mb4_unicode_ci	
6	imap	VARCHAR	100	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	NULL		utf8mb4_unicode_ci	
7	portimap	VARCHAR	5	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	NULL		utf8mb4_unicode_ci	
8	smtp	VARCHAR	100	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	NULL		utf8mb4_unicode_ci	
9	portsmtp	VARCHAR	5	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	NULL		utf8mb4_unicode_ci	
10	certificadoimap	VARCHAR	10	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	NULL		utf8mb4_unicode_ci	
11	certificadosmtp	VARCHAR	10	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	NULL		utf8mb4_unicode_ci	
12	created_at	TIMESTAMP		<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	NULL			
13	updated_at	TIMESTAMP		<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	NULL			
14	last_search	DATETIME		<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	NULL			
15	unseen_only	TINYINT	1	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	NULL			

Figura 26 Colunas do banco de dados da tabela channel_configs

Outra alteração no banco de dados foi na tabela ‘messages’, que armazena todas as mensagens que são transferidas de um usuário para outro. Nesta tabela, foram adicionados alguns campos necessários para o e-mail, como o assunto do e-mail, o corpo do e-mail, e-mails em cópia, em cópia oculta, destinatário, anexos, entre outros.

A imagem abaixo mostra uma parte desses campos na tabela ‘messages’.

#	Nome	Tipo de dados	Tamanho/It...	Unsign...	Permitir...	Zero fill	Padrão	Comentário	Colaço	Expressão
7	uid_email	INT	11	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	NULL			
8	body_plain	LONGTEXT		<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	Nenhum padrão		utf8mb4_unicode_ci	
9	attachments	VARCHAR	255	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	NULL		utf8mb4_unicode_ci	
10	from_email	VARCHAR	255	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	NULL		utf8mb4_unicode_ci	
11	from_personal	VARCHAR	255	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	NULL		utf8mb4_unicode_ci	
12	datesend	DATE		<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	NULL			
13	token	VARCHAR	255	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	NULL		utf8mb4_unicode_ci	
14	uid	VARCHAR	255	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	NULL		utf8mb4_unicode_ci	
15	contact_uid	VARCHAR	255	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	NULL		utf8mb4_unicode_ci	
16	contact_name	VARCHAR	255	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	NULL		utf8mb4_unicode_ci	
17	contact_type	VARCHAR	255	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	NULL		utf8mb4_unicode_ci	
18	message_body	LONGTEXT		<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	Nenhum padrão		utf8mb4_unicode_ci	
19	cc	LONGTEXT		<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	Nenhum padrão		utf8mb4_unicode_ci	
20	bcc	VARCHAR	255	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	NULL		utf8mb4_unicode_ci	
21	subject	LONGTEXT		<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	Nenhum padrão		utf8mb4_unicode_ci	
22	message_body_html	LONGTEXT		<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	Nenhum padrão		utf8mb4_unicode_ci	
23	message_body_b64	LONGTEXT		<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	Nenhum padrão		utf8mb4_unicode_ci	
24	email_message_id	VARCHAR	255	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	NULL		utf8mb4_unicode_ci	
25	in_reply_to	VARCHAR	255	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	NULL		utf8mb4_unicode_ci	

Figura 27 Colunas do banco de dados da tabela chat_history

5.3.3 Controllers

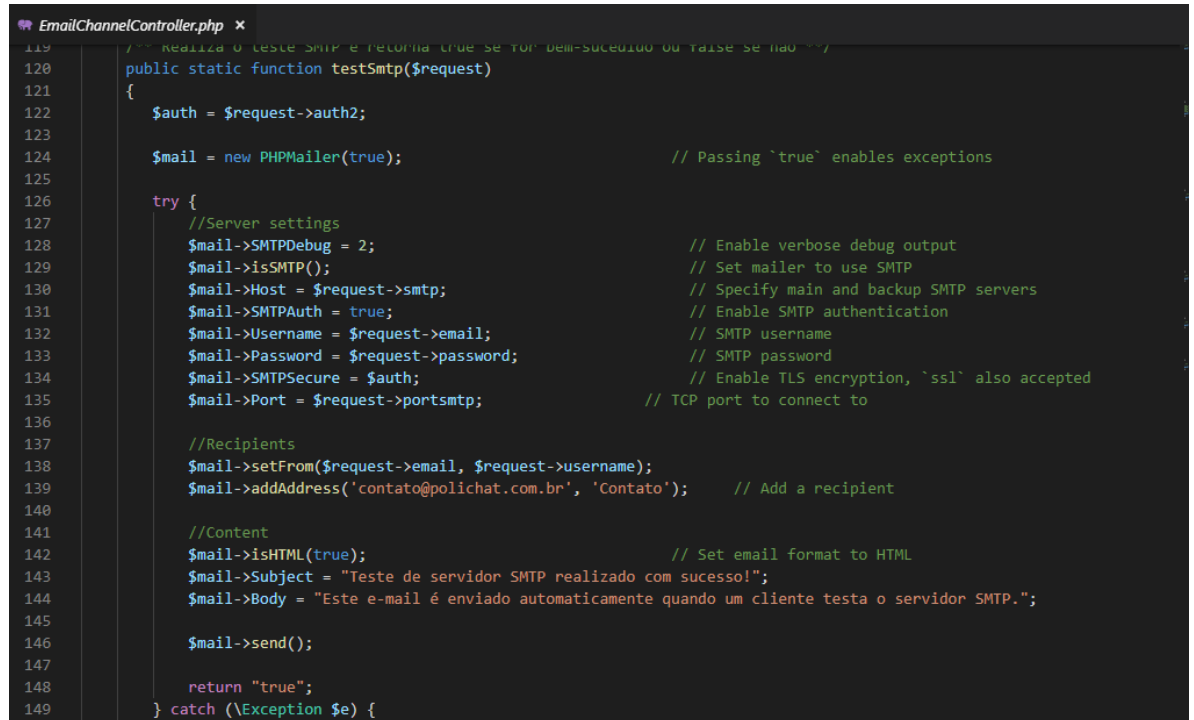
Para fazer com que tudo funcione conforme desejado, precisamos de Controllers.

O Controller criado foi o EmailChannelController, que possui métodos para:

- Entrar na página de criação, editar e excluir um canal do tipo e-mail
- Armazenar as configurações cadastradas para o e-mail
- Testar a conexão IMAP e SMTP
- Receber e-mails a partir da conexão IMAP
- Receber e-mails a partir de uma rotina de CRON que foi criada para buscar e-mails de 5 em 5 minutos
- Fazer as rotinas de recebimento de e-mail (quando recebermos um e-mail, o que desejamos fazer? Adicionar no banco de dados de qual forma? Em quais tabelas?). Tudo isso deve ser e foi programado.

- Enviar um e-mail (escrever as configurações, como: para quem este e-mail se destina, se possui um anexo ou não, se possui pessoas em cópia ou não, se possui cópia oculta, etc)

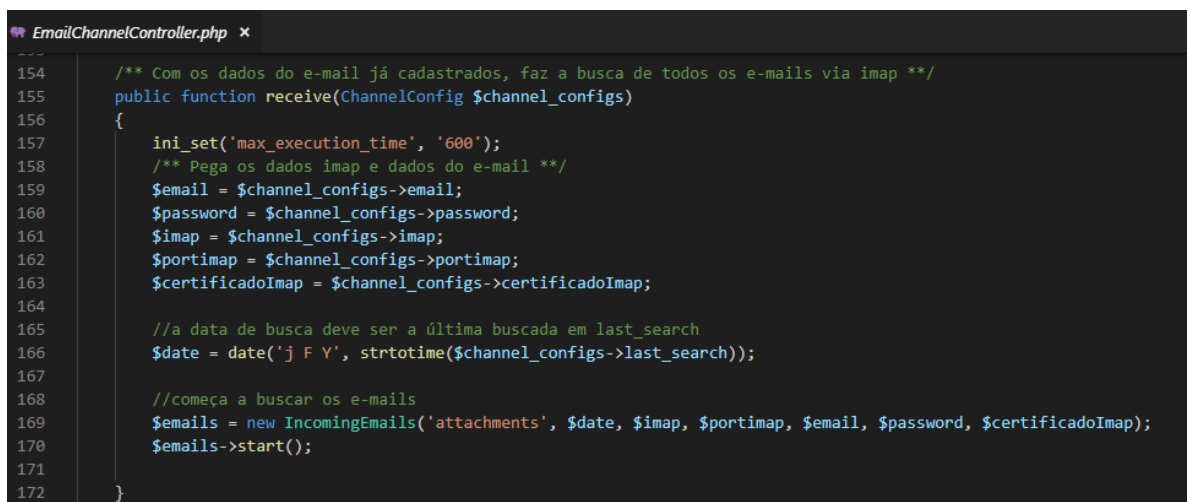
As imagens abaixo mostram alguns desses métodos do EmailChannelController:



```

119  // Realiza o teste SMTP e retorna true se foi bem-sucedido ou false se não
120  public static function testSmtp($request)
121  {
122      $auth = $request->auth2;
123
124      $mail = new PHPMailer(true);           // Passing 'true' enables exceptions
125
126      try {
127          //Server settings
128          $mail->SMTPDebug = 2;               // Enable verbose debug output
129          $mail->isSMTP();                     // Set mailer to use SMTP
130          $mail->Host = $request->smtp;         // Specify main and backup SMTP servers
131          $mail->SMTPAuth = true;              // Enable SMTP authentication
132          $mail->Username = $request->email;    // SMTP username
133          $mail->Password = $request->password; // SMTP password
134          $mail->SMTPSecure = $auth;           // Enable TLS encryption, 'ssl' also accepted
135          $mail->Port = $request->portsmtp;     // TCP port to connect to
136
137          //Recipients
138          $mail->setFrom($request->email, $request->username);
139          $mail->addAddress('contato@polichat.com.br', 'Contato'); // Add a recipient
140
141          //Content
142          $mail->isHTML(true);                 // Set email format to HTML
143          $mail->Subject = "Teste de servidor SMTP realizado com sucesso!";
144          $mail->Body = "Este e-mail é enviado automaticamente quando um cliente testa o servidor SMTP.";
145
146          $mail->send();
147
148          return "true";
149      } catch (\Exception $e) {
  
```

Figura 28 Função responsável por testar o SMTP



```

154  /** Com os dados do e-mail já cadastrados, faz a busca de todos os e-mails via imap */
155  public function receive(ChannelConfig $channel_configs)
156  {
157      ini_set('max_execution_time', '600');
158      /** Pega os dados imap e dados do e-mail */
159      $email = $channel_configs->email;
160      $password = $channel_configs->password;
161      $imap = $channel_configs->imap;
162      $portimap = $channel_configs->portimap;
163      $certificadoImap = $channel_configs->certificadoImap;
164
165      //a data de busca deve ser a última buscada em last_search
166      $date = date('j F Y', strtotime($channel_configs->last_search));
167
168      //começa a buscar os e-mails
169      $emails = new IncomingEmails('attachments', $date, $imap, $portimap, $email, $password, $certificadoImap);
170      $emails->start();
171
172  }
  
```

Figura 29 Função responsável por gerenciar o recebimento de um e-mail

```

EmailChannelController.php x
169 $emails = new IncomingEmails( $attachments, $date, $imap, $portimap, $email, $password, $certificadoimap);
170 $emails->start();
171
172 }
173
174 /** Requisição feita via cron fará com que os e-mails de todos os customers sejam lidos de forma automática */
175 public function receiveEmailsCron()
176 {
177     ini_set('max_execution_time', '600');
178
179     //Busca todos os channel_customers que possuem e-mail
180     $channel_customers = ChannelCustomer::whereNotNull('email')->get();
181
182     //percorre cada um deles, busca os e-mails chamando a função receive()
183     foreach($channel_customers as $ch) {
184         //verifica se existe o channelConfig deste canal. Caso contrário, ignorar.
185         $channelConfig = ChannelConfig::where('channel_id', $ch->id)->first();
186         if (!is_null($channelConfig)) {
187             $this->receive($channelConfig);
188         }
189     }
190 }

```

Figura 30 Função responsável por gerenciar a rotina de CRON

```

EmailChannelController.php x
193 public function receivingEmail($channel, $email, $i=0)
194 {
195     if ($email == null ) {
196         return ;
197     }
198
199     $channel_customer = ChannelCustomer::where('email', $channel)->first();
200     $message_dir="i";
201
202     /** Busca um contato com este e-mail com este customer_id */
203     $contact = Contact::where([
204         ['customer_id', '=', $channel_customer->customer_id],
205         ['email', '=', $email->from->email]
206     ])->first();
207
208     /** Verifica se o contato não existe */
209     if( $contact == null ) {
210         $contact = new Contact();
211         $contact->customer_id = $channel_customer->customer_id;
212         $contact->channel_id = $channel_customer->id;
213         $contact->category_id = '1';
214         $contact->name = ($email->from->personal == 'undefined' ? $email->from->email : $email->from->personal);
215         $contact->email = $email->from->email;
216         $contact->phone = $email->from->email;
217         $contact->type = 'user';
218         $contact->user_id = $contact->assignUser2($channel_customer->customer_id); // Operador atribuído para este cha
219         $contact->portfolio_user_id = $contact->assignUser2($channel_customer->customer_id); // Operador atribuído par
220         $contact->chat_read = false; // Email novo (Não lido)
221         $contact->status = 1; // Chat em andamento
222         $contact->new = true; // Para posteriormente fazer a verificação do contato
223         $contact->origin = $message_dir ?? null;

```

Figura 31 Função que dita o que deve ser feito ao receber um e-mail

5.3.4 Alterações no Chat

Até o momento, o Layout do chat não foi alterado. É, na verdade, a única parte que falta para que este recurso seja utilizado pelo usuário final. Porém, através do chat, que antes era apenas do Whatsapp, conseguimos receber e enviar e-mails.

A ideia é que possamos anexar, enviar e-mails em cópia, em cópia oculta, encerrar o chamado através do email, entre outros recursos. Até o momento, todas essas funcionalidades foram testadas, estamos apenas aguardando alguém responsável pelo design visual do chat fazer as alterações necessárias.

Conclusão

Trabalhar como desenvolvedor em uma Startup é algo bastante desafiador. Todos os recursos desenvolvidos até o momento, a meu ver, são novidades. Não temos resposta para tudo, devemos pesquisar e aprender cada vez mais, e isso motiva qualquer programador. Nem sempre temos respostas em livros ou em foruns da Internet, então temos que colocar a nossa lógica em prática e a nossa capacidade de criatividade para resolver os problemas demandados pela empresa.

Precisamos estar afiados com a documentação do Laravel, do PHP e das tecnologias utilizadas, para não perdermos muito tempo na hora de implementar os novos recursos.

Em uma startup, bons programadores são essenciais para o desenvolvimento de novos recursos.

Com certeza este estágio vem me agregando bastante em termos de conhecimento de empresa, conhecimentos práticos em programação em geração (banco de dados, frontend, backend, desenvolvimento baseado em testes, entre outros), conhecimentos de como uma empresa se comporta no mercado e como ela funciona internamente.

No quesito técnico, posso destacar uma evolução no meu conhecimento em git (utilizamos o Bitbucket). Comandos como 'git pull', 'git commit', 'git push', 'git branch', 'git checkout' são utilizados todos os dias na empresa para mantermos diferentes versões do software em diferentes ambientes de desenvolvimento e armazenarmos na nuvem as alterações.

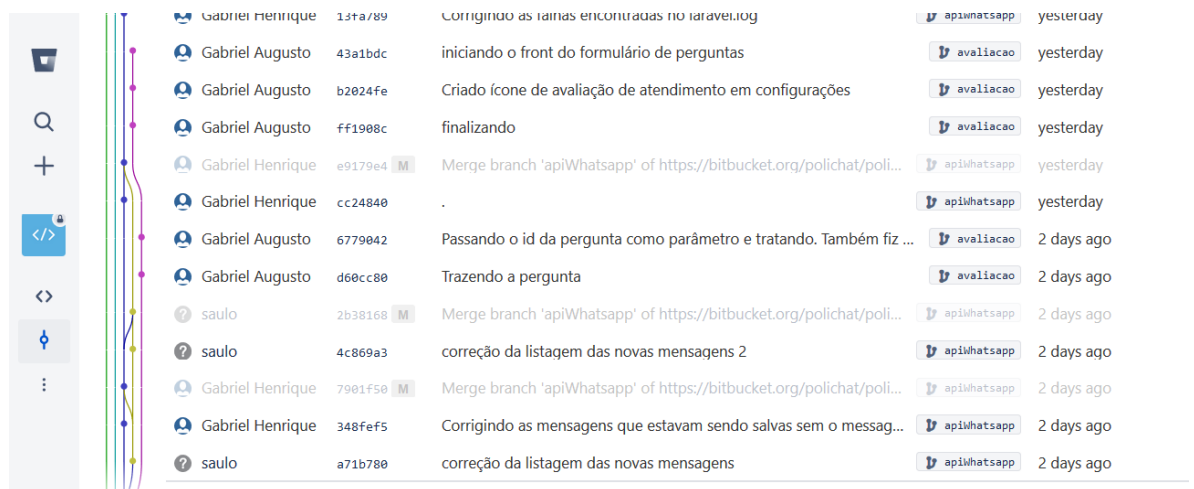


Figura 32 Commits do Bitbucket

Também destaco o uso do framework Laravel, pois o utilizo todos os dias. Afinal, o sistema foi todo desenvolvido utilizando ele.

Utilizamos bastante um gerenciador de projetos, o Trello (<http://www.trello.com>), para que a equipe toda tenha uma visão geral do que deve ser implementado, o que está em fila e o que já foi concluído.

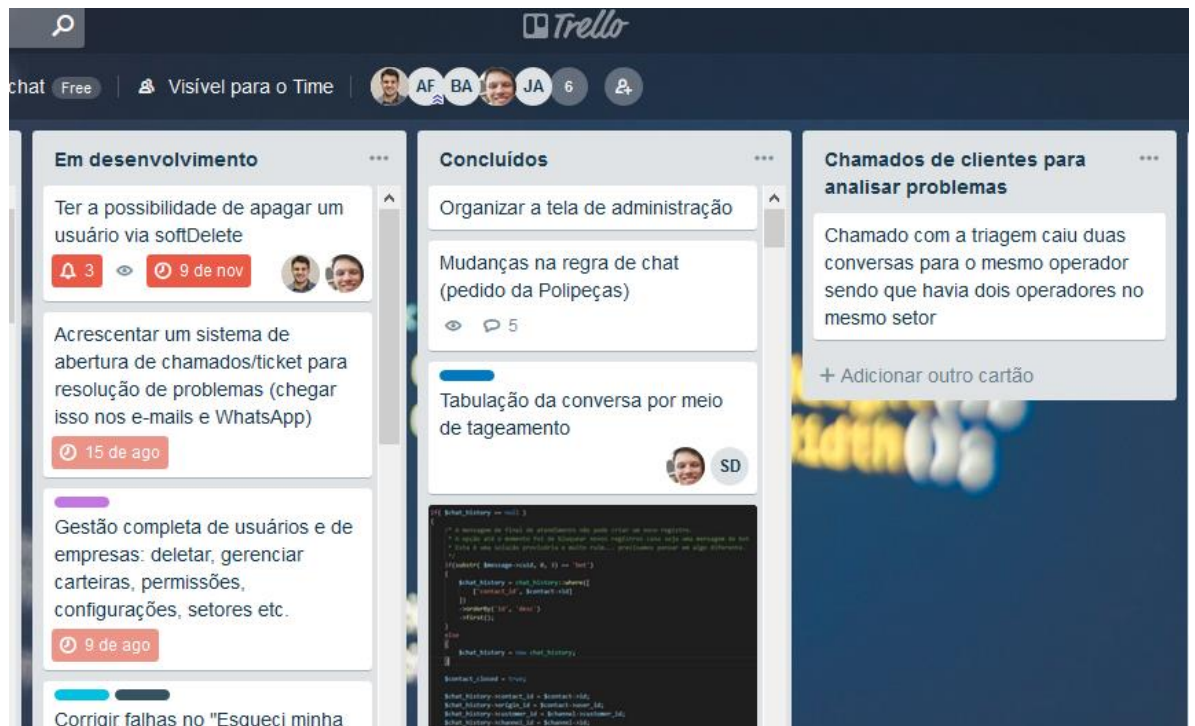


Figura 33 Roadmap da Polichat no Trello

Referências

<http://globalad.com.br/blog/o-que-sao-breadcrumbs/>
<https://canaltech.com.br/software/o-que-e-api/>