UNIVERSIDADE ESTADUAL PAULISTA "JÚLIO DE MESQUITA FILHO"
FACULDADE DE CIÊNCIAS - CAMPUS BAURU
DEPARTAMENTO DE COMPUTAÇÃO
BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

BARBARA CARVALHO SILVA

DASHBOARD MODULAR PARA CONTROLE DE PROJETOS E AMBIENTE

BAURU
2018

BARBARA CARVALHO SILVA

DASHBOARD MODULAR PARA CONTROLE DE PROJETOS E AMBIENTE

Trabalho de Conclusão do Curso de Bacharelado em Ciência da Computação apresentado ao Departamento de Computação da Faculdade de Ciências da Universidade Estadual Paulista "Júlio de Mesquita Filho" – UNESP, Câmpus de Bauru.
Orientador: Profa. Associada Roberta Spolon

BAURU
2018

BARBARA CARVALHO SILVA

DASHBOARD MODULAR PARA CONTROLE DE PROJETOS E AMBIENTE

Trabalho de Conclusão do Curso de Bacharelado em Ciência da Computação apresentado ao Departamento de Computação da Faculdade de Ciências da Universidade Estadual Paulista "Júlio de Mesquita Filho" – UNESP, Câmpus de Bauru.

Aprovado em 13/11/2018.

BANCA EXAMINADORA

Orientadora: Profa. Associada Roberta Spolon
Departamento de Computação
Faculdade de Ciências
UNESP - BAURU

Profa. Dra. Simone das Graças Domingues Prado
Departamento de Computação
Faculdade de Ciências
UNESP - BAURU

Profa. Dra. Andréa Carla Gonçalves Viana
Departamento de Computação
Faculdade de Ciências
UNESP - BAURU

AGRADECIMENTOS

Agradeço a minha orientadora Profa. Associada Roberta Spolon e a todos os professores que tive durante essa trajetória, pela orientação, incentivo e ensinamentos que me proporcionaram chegar ao final do curso.

A minha família que me apoiou durante todas as minhas escolhas.

A Matheus Coelho Solha pelo apoio, paciência, companhia e ajuda em todos os momentos.

A Gabriel Oliveira por todos os ensinamentos durante os últimos anos e no auxílio de etapas deste projeto.

A todos os amigos que fiz durante o curso que me apoiaram e tornaram esses anos inesquecíveis.

A todos que trabalham comigo, principalmente meus chefes, Fernando Hideo e Daniel Polito, pela compreensão e toda a ajuda durante minha trajetória.

"Quando a vida te decepciona, qual é a solução?
Continue a nadar! Continue a nadar!
Continue a nadar, nadar, nadar...
Para achar a solução, nadar!"
(Dori)

RESUMO

Todos os dias há uma sobrecarga de informações e essa enorme quantidade muitas vezes gera uma interpretação equivocada ou pode ocorrer até mesmo o seu esquecimento, problema que pode ser preocupante no futuro, pois sempre há dados necessários que precisam de extrema atenção. Para facilitar a manipulação e apresentação dos dados, pode-se utilizar um *dashboard*, ou seja, um painel que apresenta informações acerca de indicadores e métricas. O sistema de *dashboard* desenvolvido neste projeto tem a capacidade de interagir com algumas interfaces de programação de aplicativo, captando as informações mais relevantes e dispondo-as de uma maneira de fácil interpretação. O conjunto de informações disponíveis permite que seus usuários controlem projetos em que estão trabalhando e também aspectos do ambiente, como música, temperatura e umidade. Para tal controle, são utilizadas plataformas como *Github*, *Slack*, *Spotify* e *SInteg*, sendo este último a junção de sensores de temperatura, umidade, luminosidade e sensor magnético de portas. Com o controle do ambiente e acesso às informações mais relevantes sobre o atual projeto em desenvolvimento, pode-se tornar o trabalho mais produtivo, agradável e organizado.

Palavras-chave: *Dashboard*, Controle de projetos, Interface de programação de aplicativo.

ABSTRACT

Every day we are overloaded with information and this huge amount often generates misinterpretation or even forgetfulness, a problem that can be worrying in the future, because there is always important data that need to be dealed with extreme attention. To facilitate data manipulation and presentation, a dashboard can be used; a dashboard is a panel that presents information about indicators and metrics. The dashboard system developed in this project has the ability to interact with some application programming interfaces, capturing the most relevant information and arranging them in a way that is easy to interpret. The group of information available allows users to control the project they are working on and also control the workplace, through plataforms such as Github, Slack, Spotify and SInteg, being the latter the juncture of temperature, humidity, luminosity sensors and magnetic door sensor. Once with the workplace control and access to the most relevant information about the current project being developed, the work can be more productive, pleasant and organized.

Keywords: Dashboard. Project Control. Application Programming Interface.

LISTA DE FIGURAS

Figura 1 – *Dashboard* Operacional. 13
Figura 2 – *Dashboard* de Infraestrutura. 14
Figura 3 – Fluxo de Requisições HTTP 17
Figura 4 – Fluxo de Autenticação OAuth. 19
Figura 5 – URL de Autorização OAuth. 21
Figura 6 – Dados para obter o *access token*. 21
Figura 7 – Dashboard Spatie 23
Figura 8 – HTML *Tags* 24
Figura 9 – Página com CSS. 25
Figura 10 – Evento em Javascript. 25
Figura 11 – Diretiva *v:on* utilizada em um botão. 26
Figura 12 – *Tiles* criados com Bulma.io. 27
Figura 13 – Redes sociais mais utilizadas no mundo. 32
Figura 14 – Verificação da Versão do Node.js. 34
Figura 15 – Instalação do Express.js. 34
Figura 16 – Começando um Servidor com Node.js e Express.js. 34
Figura 17 – Escopos permitidos pelo *App* do *Slack*. 35
Figura 18 – Caminhos do servidor a partir da porta pré definida. 36
Figura 19 – Requisição para obter o *access token Slack*. 37
Figura 20 – Fluxo de requisições entre o servidor e APIs. 37
Figura 21 – Métodos específicos para canais do *Slack*. 38
Figura 22 – Requisição para obter dados do *Slack*. 39
Figura 23 – Requisição para obter dados sobre um projeto do *Github*. 40
Figura 24 – Requisição para obter dados provindos do sensor de luminosidade do *SInteg*. 40
Figura 25 – Definição de cores e estilos. 41
Figura 26 – Protótipo do módulo *SInteg*. 42
Figura 27 – Estrutura dos componentes do sistema. 42
Figura 28 – Redirecionamento para página de autorização do *Slack*. 44
Figura 29 – Fluxo de dados e requisições entre API, servidor e *frontend*. 44
Figura 30 – Requisição de autorização para o usuário. 45
Figura 31 – Requisição do servidor para a API. 45
Figura 32 – *Card* com as informações do projeto do *Github*. 46
Figura 33 – *Card* com as informações da milestone de um projeto do *Github*. 46
Figura 34 – *Card* com as informações do *Slack*. 47
Figura 35 – Função que obtém o nome do usuário a partir de seu ID no *Slack*. 48

Figura 36 – *Card* com as informações do *Spotify*. 49
Figura 37 – *Script* que insere o *player* do *Spotify*. 49
Figura 38 – Mensagem para o usuário trocar o dispositivo de áudio. 50
Figura 39 – Seleção de dispositivos do *Spotify*. 50
Figura 40 – URLs para requisições do *SInteg*. 51
Figura 41 – *Card* com as informações do *SInteg*. 51
Figura 42 – Função retirada da documentação da API do *Facebook*. 52
Figura 43 – Informação sobre inutilização da API do *Instagram*. 52

LISTA DE ABREVIATURAS E SIGLAS

TI	Tecnologia da Informação
IOT	*Internet of Things* - Internet das Coisas
API	*Application Programming Interface* - Interface de Programação de Aplicativos
KPI	*Key Performance Indicator* - Indicador Chave de Desempenho
SPA	*Single Page Application* - Aplicativo de Única Página
HTTP	*Hypertext Transfer Protocol* - Protocolo de Transferência de Texto
HTML	*Hypertext Markup Language* - Linguagem de Marcação de Texto
URL	*Uniform Resource Locator* - Localizador Uniforme de Recursos
SDK	*Software Development Kit* - Kit de Desenvolvimento de *Software*
CSS	*Cascading Style Sheets* - Folha de Estilo em Cascatas
SASS	*Syntactically Awesome Style Sheets*
DOM	*Document Object Model* - Modelo de Objeto de Documento
NPM	*Node Package Manager* - Gerenciador de Pacotes Node
LDR	*Light Dependent Resistor* - Resistor Dependente de Luz

SUMÁRIO

1 INTRODUÇÃO 12
1.1 Justificativa 14
1.2 Objetivos 15
1.2.1 Objetivos Gerais 15
1.2.2 Objetivos Específicos 15
1.3 Organização do trabalho 15

2 FUNDAMENTAÇÃO TEÓRICA 16
2.1 *Dashboards* 16
2.2 *Websites* 16
2.3 Servidores *Web* 17
2.4 API 18
2.5 OAuth 2.0 19
2.5.1 Tipos de garantias de autorização 20
2.5.2 *Authorization Code* 20

3 METODOLOGIA 23
3.1 Métodos e Etapas 23
3.2 Tecnologias e Ferramentas 24
3.2.1 HTML 24
3.2.2 CSS e SASS 24
3.2.3 Javascript 25
3.2.4 Vue.js 26
3.2.5 Bulma.io 27
3.2.6 Node.js e Express.js 28
3.2.7 *Web Playback* SDK 28
3.2.8 Git 28

4 DESENVOLVIMENTO 30
4.1 Estudo e Definição das APIs 30
4.1.1 *Github* 30
4.1.2 *Slack* 31
4.1.3 *Facebook e Instagram* 31
4.1.4 *Spotify* 32
4.1.5 *SInteg* 32
4.2 Desenvolvimento do Servidor 33

4.2.1 Instalação e Configuração do Node.js e Express.js 33
4.2.2 Cadastro das aplicações 34
4.2.3 Requisições de Autorização 36
4.2.4 Requisições para obtenção de dados 38
4.3 Desenvolvimento dos módulos 41
4.4 Integração do Servidor com o *Dashboard* 43
4.4.1 *Github* 45
4.4.2 *Slack* 46
4.4.3 *Spotify* 48
4.4.4 *SInteg* 50
4.4.5 Problemas Enfrentados 51

5 CONCLUSÃO 53
5.1 Trabalhos Futuros 53

REFERÊNCIAS 55

1 INTRODUÇÃO

Atualmente vive-se na era da informação, onde houve grande avanço tecnológico advindo da terceira revolução industrial, a qual possibilitou invenções como o microprocessador, a rede de computadores, a fibra ótica, entre outros. Estas criações geraram uma grande quantidade de informações.

> "Informação é a resultante do processamento, manipulação e organização de dados, de tal forma que represente uma modificação (quantitativa ou qualitativa) no conhecimento do sistema (humano, animal ou máquina) que a recebe."(SERRA, 2007, p.98)

Todos os dias há uma sobrecarga de informações e essa enorme quantidade muitas vezes gera uma interpretação equivocada ou pode ocorrer até mesmo o seu esquecimento, problema que pode ser preocupante no futuro, pois sempre há dados necessários que precisam de extrema atenção.

De acordo com Coadic (1996), o valor da informação varia conforme o indivíduo, as necessidades e o contexto em que é produzida e compartilhada. Uma informação pode ser altamente relevante para um indivíduo e a mesma informação pode não ter significado algum para outro indivíduo.

O atual cenário corporativo é extremamente complexo e competitivo, e este excesso de dados precisa ser filtrado de acordo com a necessidade de cada empresa, sempre com a devida cautela para que todos os dados necessários sejam coletados e tratados da maneira correta para que os mesmos possam ser estruturados, interpretados e apresentados de forma concreta e precisa.

Para facilitar a manipulação e apresentação dos dados, pode-se utilizar um *dashboard*, ou seja, um painel que apresenta informações acerca de indicadores e métricas.

Os *dashboards* mais conhecidos são os utilizados em carros. O mesmo serve para informar ao motorista sobre a situação do veículo, sendo eles velocidade, rotação do motor, nível de gasolina, nível de óleo, entre outros. Este painel ajuda o motorista a tomar decisões de acordo com a urgência da informação disponibilizada.

Além de sua utilização em carros, também aparecem na área de marketing, recursos humanos, logística, financeira, e diversas outras.

Os painéis empresariais tem o mesmo objetivo de um painel para carros, sendo uma forma para os gestores e funcionários terem conhecimento rápido dos dados importantes para a empresa, auxiliando na tomada de decisões.

O *dashboard* operacional, por exemplo, monitora processos de negócios que mudam frequentemente, os seus indicadores de desempenho (KPI - *Key Performance Indicator*) garantem o bom desempenho da operação, identificando gargalos e etapas críticas da operação. Neste painel são apresentados os dados que auxiliam os analistas a corrigir erros e falhas nos processos. A Figura 1 apresenta um exemplo do modelo descrito.

Figura 1 – *Dashboard* Operacional.

Fonte: OP Services (2017)

As possibilidades de *dashboards* para a área de TI são muito grandes, podem ser analisadas informações sobre o desempenho de *datacenters*[1], links de internet, equipamentos do ambiente de computação, produtividade das equipes, e várias outras características que são indicadores de desempenho.

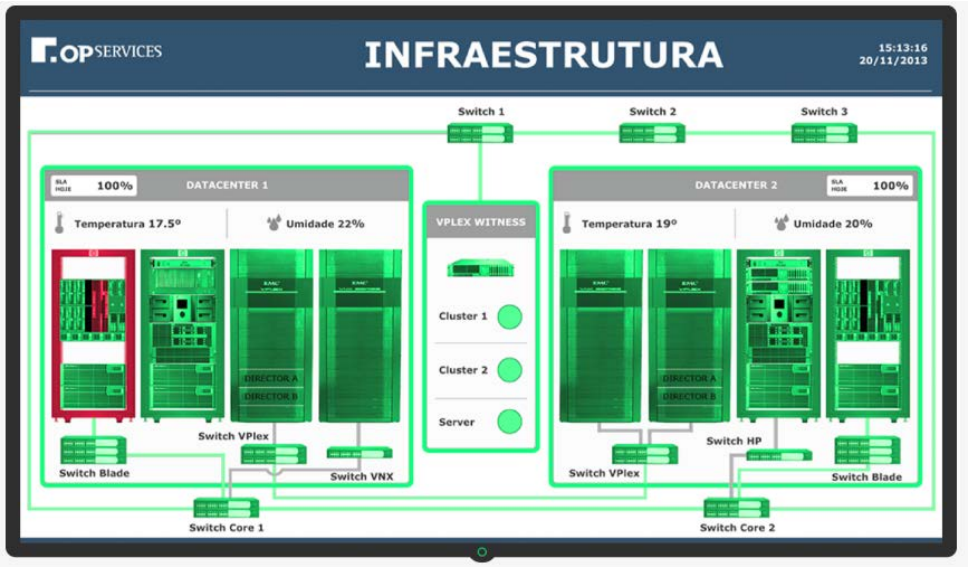
Para lidar com o controle de informações de *data centers*, foi desenvolvido um painel que disponibiliza informações sobre a temperatura, umidade e *switches* [2]. Caso ocorra algum problema, como o superaquecimento de um servidor, o mesmo pode ser resolvido em pouco tempo, garantindo mais segurança à todo o projeto. A Figura 2 apresenta o painel descrito.

Para lidar com os diversos desafios dos negócios e atingir as metas desejadas, pode-se escolher diferentes tipos de painéis de acordo com a sua necessidade, garantindo melhor interpretação dos dados e uma maior produtividade e lucro.

[1] Ambiente projetado para abrigar servidores e outros componentes como sistemas de armazenamento de dados e ativos de rede. O objetivo principal de um Data Center é garantir a disponibilidade de equipamentos que rodam sistemas cruciais para o negócio de uma organização. (TELECORP, 2017)

[2] Equipamento que possibilita a conexão de computadores em uma rede local. (IN8, 2018)

Figura 2 – *Dashboard* de Infraestrutura.



Fonte: OPServices (2016)

1.1 Justificativa

A utilização de um modelo de *dashboard* facilita o acompanhamento das operações de uma empresa, tornando as decisões mais transparentes e proporcionando maior comunicação, gerando maior integração e envolvimento de toda a equipe.

> "O cérebro humano tem mais capacidade de visualizar todas as fontes de informação inter-relacionados em conjunto, desta forma, é possível entender a importância e significado global de um conjunto de informação com maior precisão. Ele também permite comparações mais fáceis e rápidas entre os diferentes tipos de gráficos, bem como a identificação de tendências e relações dentro do conjunto de dados global, levando a uma percepção mais profunda". (CARVALHO, 2016)

Os *dashboards* existentes para a área de computação permitem o controle específico de alguns segmentos, principalmente para a infraestrutura. Porém, não há nenhum painel que permite o controle de projetos e de ambiente.

O sistema desenvolvido é modular, garantindo a integração de diversas funções que auxiliam em diferentes aspectos. A disponibilidade dos dados facilita na gestão e controle do projeto e também garante um ambiente de trabalho mais agradável, proporcionando maior produtividade.

1.2 Objetivos

1.2.1 Objetivos Gerais

O objetivo geral desse projeto é desenvolver um sistema de *dashboard* modular para controle de projetos e ambiente, garantindo um melhor gerenciamento das informações, ampliando a visão sobre o que está sendo feito e o que ainda deve ser realizado, também possibilitar um ambiente mais agradável para aumentar a produtividade dos usuários.

1.2.2 Objetivos Específicos

a) Analisar os modelos de *dashboards* já existentes;

b) Estudar e definir os principais módulos a serem desenvolvidos;

c) Estudar métodos para garantir que a interface contenha elementos de fácil acesso e seja amigável para com o usuário;

d) Planejar a estrutura do sistema;

e) Estudar e coletar os dados que deverão ser exibidos em cada módulo;

f) Implementar o protótipo e integrar com os dados coletados das APIs;

g) Integrar o protótipo com os módulos baseados em IoT[3].

1.3 Organização do trabalho

Esta monografia foi organizada da maneira que segue.

No capítulo 2, encontra-se a fundamentação teórica necessária para o entendimento e a execução deste trabalho, abordando conceitos como servidores web, APIs e o protocolo OAuth 2.0.

O capítulo 3 apresenta a metodologia, ferramentas e tecnologias utilizadas durante o desenvolvimento do projeto.

O capítulo 4 apresenta as etapas do desenvolvimento do projeto, sendo estes o estudo e construção dos módulos e servidor e a integração dos mesmos.

Por fim, no capítulo 5, têm-se as considerações finais sobre o projeto e os trabalhos futuros.

[3] Projeto de conclusão de curso desenvolvido por Matheus Coelho Solha que consiste na construção de módulos baseados em IoT para a coleta de dados e gerenciamento de informações. O mesmo contém sensores que disponibilizarão dados para o *dashboard*.

2 FUNDAMENTAÇÃO TEÓRICA

2.1 *Dashboards*

Dashboards são painéis visuais que apresentam um conjunto de informações importantes para alcançar objetivos e metas traçadas, auxiliando na tomada de decisões.

O objetivo dos *dashboards* é possibilitar o monitoramento dos resultados de uma empresa distribuídos em diversos indicadores. Para definir as métricas e indicadores é preciso compreender as necessidades da empresa. (E.MIX, 2016)

O ponto principal em um painel deve ser a sua capacidade de transmitir a informação de forma rápida e eficiente.

Existem três tipos de *dashboards* para suprir as necessidades de monitoramento dos resultados:

a) **Operacionais:** contribuem para monitorar níveis de operação de analistas;

b) **Táticos:** permitem ter uma visão de como estão as operações de acordo com as estratégias estabelecidas;

c) **Estratégicos:** monitoram os indicadores chave de desenvolvimento e analisam as metas estabelecidas.

É comum que os painéis sejam exibidos em telas grandes de LCD em diversas localidades das empresas, para que seja possível acompanhar os indicadores de desempenho e para que todos os profissionais se sintam mais integrados aos processos da organização. (OP SERVICES, 2017)

2.2 *Websites*

A palavra *website* faz referência a uma página ou a um agrupamento de páginas relacionadas entre si, que residem em um espaço virtual da internet, sendo este disponibilizado através de um servidor, acessível através de um determinado endereço. Há diversos tipos de *websites* e cada um possui um objetivo, de acordo com o público ao qual é direcionado. (SIGNIFICADOS, 2011)

Dentro de um *website* podem ser colocados textos, imagens, vídeos ou animações e as páginas são carregadas através do protocolo de rede HTTP (*Hypertext Transfer Protocol*), sendo visualizadas através de um navegador da preferência do usuário.

Existem dois tipos de *websites*: estático e dinâmico. Um *website* composto apenas de código HTML e conteúdo audiovisual é classificado como estático, pois não há armazenamento de dados e seu conteúdo não é modificado. Já um *website* dinâmico possui muito mais funcionalidades, uma vez que possui a capacidade de alterar o seu conteúdo através do carregamento de dados vindos de um banco de dados residente no servidor. (W/MORAIS, 2016)

2.3 Servidores *Web*

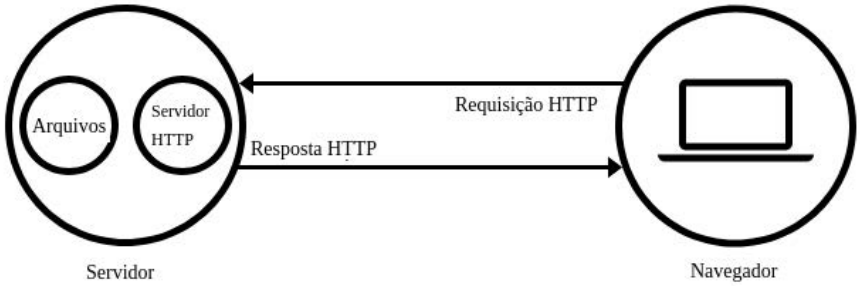
A referência a um servidor *web* pode ser tanto sobre a parte do *hardware*, do *software* ou os dois em conjunto.

Referente ao *hardware*, é um computador que armazena os arquivos componentes de sites e os entrega para o dispositivo do usuário final. Está conectado à *internet* e pode ser acessado através do seu nome de domínio.

Referente ao *software*, inclui diversos componentes que controlam como os usuários acessam os arquivos hospedados, agindo com um servidor HTTP. Um servidor HTTP é um *software* que compreende URLs, que são os endereços *web*, e HTTP, protocolo que o navegador utiliza para visualizar páginas *web*.

O navegador faz uma requisição utilizando o protocolo HTTP sempre que necessitar de algum arquivo hospedado no servidor *web*, e o dado requerido será enviado pelo servidor para o navegador via HTTP. O fluxo das requisições é apresentado na Figura 3.

Figura 3 – Fluxo de Requisições HTTP



Fonte: Mozilla (2018)

Para produzir as páginas finalizadas, o servidor de aplicações pode completar um modelo de página HTML com o conteúdo obtido de um banco de dados. Sites como o MDN ou a Wikipédia possuem milhares de páginas *web*, mas elas não são realmente documentos HTML, mas apenas alguns poucos *templates* HTML e uma gigantesca base de dados. Essa configuração agiliza e facilita o gerenciamento e a entrega do conteúdo. (MOZILLA, 2018)

2.4 API

A sigla API vem de "*Application Programming Interface*", no português "Interface de Programação de Aplicações". São um conjunto de rotinas e padrões de programação para acesso a um aplicativo de *software* ou plataforma baseado na *web*, permitindo integrar sistemas, garantindo a segurança de dados. (CANALTECH, 2017)

As APIs proporcionam a integração entre sistemas desenvolvidos em linguagens distintas de maneira ágil e segura. Em outras formas de integração de sistemas seria necessário instalar recursos compatíveis com o sistema no qual se busca efetuar a integração, gerando um grande trabalho e, consequentemente, atraso na geração de negócios e processos produtivos.

APIs são criadas quando uma empresa de *software* tem a intenção de que outros programadores de *software* desenvolvam produtos associados ao seu serviço. Muitas instituições disponibilizam seus códigos e instruções para serem integrados da maneira mais conveniente para seus usuários.

Há inúmeras vantagens na utilização de APIs:

a) **Segurança:** A maneira como a integração é feita proporciona uma troca de informações de forma muito segura, uma vez que os dados só são disponibilizados depois de se realizar uma série de requisições através de um protocolo padrão de autorização.

b) **Redução no volume de dados:** As APIs são capazes de restringir a quantidade de informações de acordo com a necessidade do sistema, o que permite que o sistema não gaste com dados desnecessários.

c) **Monitoramento de acessos:** APIs são capazes de monitorar quem acessou, quando e onde a requisição foi feita e também quais informações foram disponibilizadas. Uma característica bem importante para sistemas de comércio eletrônico ou que manipulem dados de valor.

Com o avanço da integração das informações, muitas APIs foram criadas e por mais que não sejam visíveis para os usuários, elas estão sendo utilizadas em grande parte dos sistemas.

A API do *Facebook* é a mais utilizada atualmente, e como todas as redes sociais, armazena informações que podem ser úteis para avaliar hábitos de consumo, como idade, sexo, locais frequentados, entre outras características. Essas informações podem ser fundamentais para uma estratégia de marketing digital e para área comercial de uma organização, amplificando seus resultados. (VERTIGO TECNOLOGIA, 2018)

2.5 OAuth 2.0

O *OAuth* 2.0 é um protocolo padrão de autorização que permite que usuários tenham acesso limitado a recursos de um *website* sem precisar expor suas credenciais (DIGITAL OCEAN, 2018). Como o mesmo é uma especificação, descreve alguns papéis envolvidos em uma comunicação que utiliza o protocolo:

a) ***Resource Owner*:** A entidade que controla o acesso aos recursos protegidos.

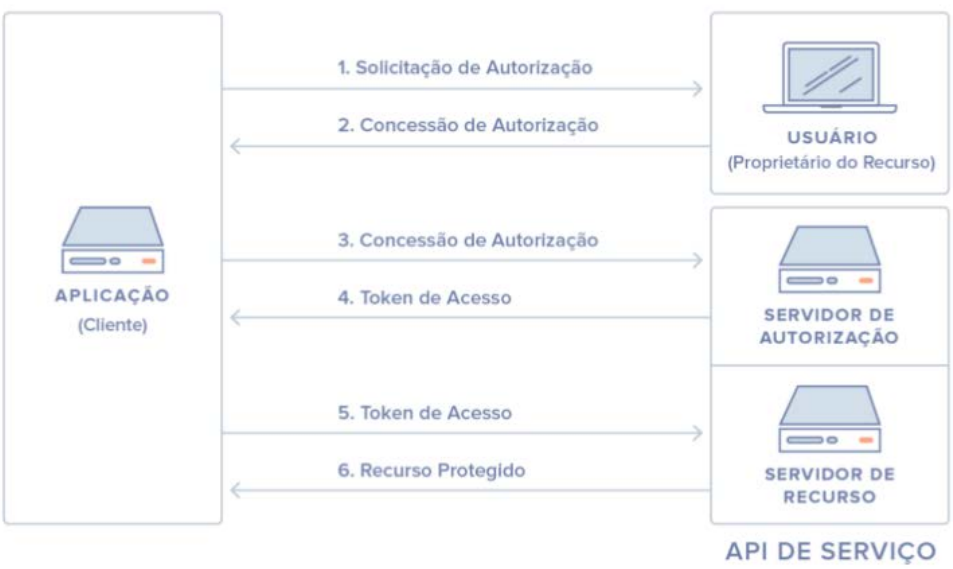
b) ***Resource Server*:** Servidor responsável por armazenar os recursos a serem acessados e informar a API que está fazendo uma requisição.

c) ***Client*:** A aplicação que solicita acesso aos recursos protegidos do *Resource Owner*;

d) ***Authorization Server*:** Servidor que gera os *tokens* de acesso, permitindo que o *Client* acesse os recursos que o *Resource Owner* permitiu, com o nível de acesso que este especificou.

A Figura 4 representa a sequência de eventos até a obtenção do *token* de acesso.

Figura 4 – Fluxo de Autenticação OAuth.



Fonte: Digital Ocean (2018)

O *Client* pede autorização ao *Resource Owner* para acessar seus recursos, quando o mesmo autorizar o acesso, o *Client* receberá uma garantia de autorização. O *Client* solicita um *access token* ao *Authorization Server*, enviando a garantia de autorização recebida anteriormente. Assumindo que o *Client* foi autorizado e que a garantia de autorização é válida, o *Authorization Server* gera um *access token* e o retorna ao *Client*. O *Client* pede acesso a um recurso protegido pelo *Resource Server*, e é autenticado através do *access*

token, se o mesmo for válido, o *Resource Server* responde à requisição do *Client* servindo o recurso solicitado.

Algumas plataformas que permitem a sincronização através do OAuth 2.0 utilizam *tokens* que expiram após um período, estes são chamados *refresh tokens*, eles possuem as informações necessárias para gerar um novo *access token*, basta fazer uma nova requisição com o mesmo. (IMASTERS, 2017)

2.5.1 Tipos de garantias de autorização

Na primeira requisição feita pelo *Client*, é retornado uma garantia de autorização, o seu tipo varia de acordo com o tipo da garantia de autorização que foi enviada na requisição. De acordo com (OAUTH, 2018), há quatro tipos de garantias de autorização:

a) ***Authorization Code*:** Utilizado em aplicações *web* que executam em servidores, usado principalmente para obter recursos de usuários de aplicações de terceiros, como por exemplo *Facebook*, *Spotify* e *Instagram*.

b) ***Implicit*:** Utilizado por SPAs (*single page applications*) que executam em *browsers*.

c) ***Resource Owner Password Credentials*:** Utilizado por aplicativos confiáveis, que são definidos através de restrições de acesso do *Resource Owner*.

d) ***Client Credentials*:** Utilizado em comunicações do tipo máquina para máquina, como por exemplo aplicações não interativas onde o *token* é emitido para o próprio aplicativo, em vez de um usuário final.

2.5.2 *Authorization Code*

O tipo de autorização utilizado neste projeto é o *Authorization Code*, sendo assim, cabe explicar mais profundamente como é feita a sua requisição e como é seu fluxo.

Para iniciar a autenticação, o *Client* deve enviar os dados do usuário à URL de autorização, a qual varia de acordo com a aplicação a ser consumida, e pode ser encontrada na documentação da aplicação em questão na seção de autorização com *OAuth*. A Figura 5 apresenta uma URL de autorização OAuth para a API do *Spotify*.

Sendo, na respectiva ordem:

- A URL para qual será feita a requisição de autenticação;

- *Response type*: especifica que sua aplicação espera receber um *Authorization Code*;

- *Client id*: o ID que você recebeu quando criou sua aplicação no domínio requisitado;

Figura 5 – URL de Autorização OAuth.

```
app.get('/spotify/auth', (req, res) => {
    var scopes = ["streaming", "user-read-birthdate", "user-read-email", "user-read-private"];
    res.redirect(
        'https://accounts.spotify.com/authorize'
        + '?response_type=code'
        + '&client_id=' + 'b35dda2c6d8844eb9ca34c7b405ef4d5'
        + (scopes ? '&scope=' + encodeURIComponent(scopes) : '') +
        '&redirect_uri=' + encodeURIComponent('http://localhost:4392/spotify/token')
    );
});
```

Fonte: elaborado pelo autor

- *Scopes*: os valores de escopo indicando quais recursos do usuário você deseja obter acesso;

- *Redirect URL*: a URL para onde o usuário será redirecionado após a autenticação ter sucesso.

Fazendo esta requisição, todos estes dados serão acoplados à sua URL e o usuário será informado de que uma API foi chamada e deseja compartilhar alguns dados. Após o consentimento do usuário, um *authorization token* será retornado e é preciso trocá-lo por um *access token*, que será utilizado em todas as requisições à API.

Para trocar o *authorization token* pelo *access token* é necessário realizar outra requisição. A Figura 6 apresenta os dados para obter o *access token* da API do *Spotify*.

Figura 6 – Dados para obter o *access token*.

```
var postConfig = {
    url: 'https://accounts.spotify.com/api/token',
    method: 'post',
    data: {
        code: req.query.code,
        redirect_uri: 'http://barbatheus.dashboard.com:4392/spotify/token',
        grant_type: 'authorization_code',
    },
    headers: {
        'Authorization': 'Basic '
            + (new Buffer(
                process.env.SPOTIFY_CLIENT_ID + ':'
                + process.env.SPOTIFY_CLIENT_SECRET
            ).toString('base64')),
        'Content-Type': 'application/x-www-form-urlencoded',
    },
};
```

Fonte: elaborado pelo autor

Abaixo são descritos os dados necessários para a realização desta requisição.

- URL: a URL para qual será feita a requisição;

- *Method*: o tipo de requisição a ser feita, que neste caso deve ser um *POST*;

- *Code*: o *authorization code* recebido da chamada de autorização feita anteriormente;

- *Redirect* URL: a URL para onde o usuário será redirecionado após a autenticação ter sucesso, a qual deve ser exatamente igual à URL passada na requisição anterior;

- *Grant type*: informa o tipo de code que está sendo mandado, neste caso é um *authorization code*;

- *Client id*: o *client id* da sua aplicação;

- *Client secret*: o *client secret* da sua aplicação;

A resposta desta requisição conterá o *access token* e pode ser utilizado em requisições para obter informações sobre o usuário, desde que ele tenha autorizado previamente.

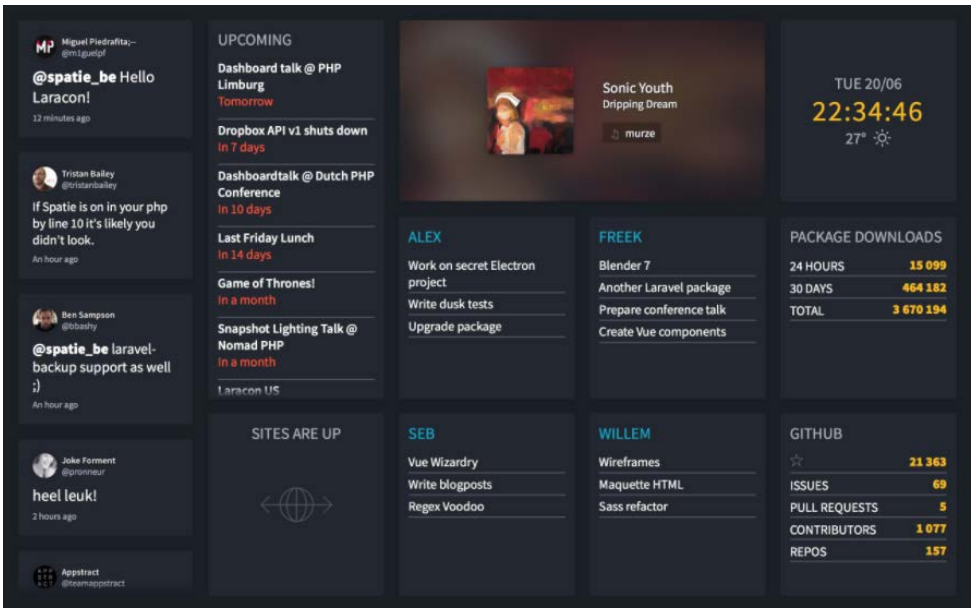
3 METODOLOGIA

3.1 Métodos e Etapas

O projeto de desenvolvimento do *dashboard* foi dividido em três etapas. A primeira foi o levantamento bibliográfico relacionado ao tema, na qual foram realizadas buscas relacionadas aos painéis já existentes, como o *dashboard* Spatie, desenvolvido por Freek Van der Herten. Concomitantemente foi realizado o estudo de APIs e a definição dos módulos a serem desenvolvidos.

A Figura 7 apresenta o *dashboard* Spatie.

Figura 7 – Dashboard Spatie



Fonte: Herten (2017)

A segunda etapa consistiu em desenvolver um servidor para que fosse possível obter os dados necessários provindos das APIs, visto que é necessário uma série de requisições e as mesmas possuem dados que devem ser guardados no *backend*, fora do alcance de usuários.

A terceira etapa foi a integração do servidor com o *frontend*, a finalização dos módulos, garantindo que os dados obtidos fossem precisos e disponibilizados em seu respectivo módulo de uma maneira agradável para com o usuário.

3.2 Tecnologias e Ferramentas

3.2.1 HTML

A linguagem utilizada atualmente para a construção da maioria das páginas *web* é HTML, ou *HyperText Markup Language* (linguagem de marcação em hipertexto). Um navegador ao ler um documento HTML, interpreta as *tags* que este documento contém para mostrar a maneira na qual os dados serão exibidos.

Como exemplificado na Figura 8, o HTML é uma linguagem de marcação, e as mesmas são feitas através de *tags*, separando a página em diferentes seções, dependendo de seu conteúdo, para gerar o que será exibido, deixando os elementos organizados para um melhor entendimento.

Figura 8 – HTML *Tags*

```
<!DOCTYPE html>

<html lang="pt-br">
<head>
    <meta charset="utf-8">
    <title>Título da página</title>
</head>
<body>
 ... aqui vai todo o codigo HTML que faz seu site...
</body>
</html>
```

Fonte: w3schools (2018)

3.2.2 CSS e SASS

CSS é uma linguagem de folhas de estilos em cascata, que age diretamente em uma página *web*, deixando-a mais agradável visualmente, desde o tamanho da fonte até na criação de figuras. É considerado uma folha de estilo em cascata porque é possível inserir uma folha dentro de outra para compor o elemento final. (HOSTINGER, 2018)

O CSS e o HTML são extremamente ligados, pois a customização da página se dá através de estilos associados às *tags*, como mostrado na Figura 9.

O Sass é uma linguagem de folhas de estilos interpretada em CSS. É uma extensão de CSS que permite o uso de variáveis, regras, *mixins* e importações. O Sass ajuda a manter as grandes folhas de estilo bem organizadas e a obter pequenas folhas de estilo rapidamente. (SASS, 2018)

Como o projeto é modular e os módulos possuem o mesmo estilo, o uso de Sass permite com que o código permaneça enxuto e organizado, uma vez que pode-se utilizar variáveis.

Figura 9 – Página com CSS.


```
<!DOCTYPE html>
<html>
<head>
<style>
body {
    background-color: lightblue;
}
h1 {
    color: white;
    text-align: center;
}
</style>
</head>
<body>

<h1>My First CSS Example</h1>
<p>This is a paragraph.</p>

</body>
</html>
```

Fonte: elaborado pelo autor

3.2.3 Javascript

Javacript é uma linguagem de programação interpretada, foi criada principalmente para ser usada no *client-side*, permitindo que os *scripts* pudessem ser rodados e interagissem com o usuário sem a necessidade de passar pelo servidor, porém também é comumente utilizada no *server-side* através de ambientes como o Node.js. (DEVFURIA, 2018)

É uma linguagem estruturada, baseada em objetos, dinâmica e suporta variáveis de inúmeros tipos e também expressões regulares. Sendo assim, é utilizado para adicionar interatividade a uma página *web*. (JAVASCRIPT INFO, 2018)

Pode-se adicionar um evento para gerar uma resposta ao se clicar em um botão e até mesmo abrir uma nova página no navegador, como mostrado na Figura 10.

Figura 10 – Evento em Javascript.

```
<!DOCTYPE html>
<html>
<body>

<p>Click the button to open slack authorize link.</p>

<button onclick="authSlack()">Try it</button>

<script>
  function authSlack() {
      window.open("https://slack.com/oauth/authorize");
  }
</script>

</body>
</html>
```

Fonte: elaborado pelo autor

3.2.4 Vue.js

Vue.js é um *framework* Javascript para construção de componentes reativos para interfaces *web*. Sua biblioteca principal é focada na parte visual e como utiliza Javascript ele é facilmente integrável com outras bibliotecas e projetos já existentes independente da linguagem de desenvolvimento. (VUEJS, 2017)

Ele é reativo, ou seja, é possível observar um objeto Javascript e suas alterações no DOM do HTML, caso o valor de uma variável seja alterado no *console*, o valor será atualizado simultaneamente na página *web*.

Possui diretivas, que são atributos especiais providos pelo mesmo, elas aplicam um comportamento especial de reatividade ao DOM renderizado. Um exemplo de diretiva é o *v-bind*, que mantém o elemento sempre atualizado em relação a propriedade que for atribuída a ele. (SIMEAO, 2017)

Outra característica do Vue.js é que ele permite a interatividade do usuário com o sistema, a qual pode ser feita através da diretiva *v-on*, por exemplo, que é utilizada para escutar eventos e pode ser acoplada a um botão, como na Figura 11, que ao clicar no botão um método será chamado e a mensagem que está escrita acima do botão será escrita da forma reversa na mesma hora em que o botão for acionado, por causa da sua reatividade.

Figura 11 – Diretiva *v:on* utilizada em um botão.

Fonte: VueJS (2017)

A componentização é outra grande vantagem do Vue.js, pois é uma abstração que proporciona a construção de aplicações de larga escala compostas por pequenos componentes, auto-contidos e frequentemente reutilizáveis, tornando o desenvolvimento

gerenciável e prático. (VUEJSBR, 2016)

Um componente consiste em partes de código que possuem um estilo ou comportamento e pode ser reutilizado em outro componente. Mesmo que a aplicação seja dividida em componentes, eles são capazes acessar informações uns dos outros.

3.2.5 Bulma.io

Bulma.io é um *framework* de código aberto baseado em *flexbox*, totalmente responsivo e modular. O seu pacote possui os elementos mais utilizados comumente, como botões, formulários, barra de progresso, sistema de *grid*, entre outras funcionalidades. (PRATES, 2016)

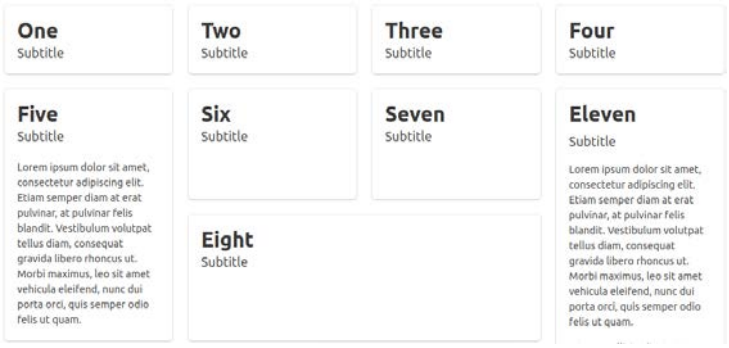
Como ele não utiliza Javascript, apenas CSS, é facilmente integrável com qualquer ambiente. Ele consiste em arquivos Sass que podem ser importados individualmente caso não seja desejado a instalação de todos os arquivos em um projeto.

Várias cores podem ser utilizadas e elas possuem modificadores já definidos, como por exemplo o *invert value*, que é calculado em uma função para garantir que caso algo seja escrito sobre a cor desejada, o texto possa aparecer sem dificuldade para leitura.

O padrão de seus elementos são para *mobile*, para cada tipo há um certo valor de pixels. As opções são: *mobile*, *tablet*, *desktop*, *widescreen* e *fullHD*. Como em vários componentes do Bulma.io, esta característica pode ser alterada facilmente adicionando uma *tag* ao elemento. (BULMA.IO, 2017)

Um exemplo de componente são os *tiles*, como o Bulma.io é baseado em *flexbox*, ele permite a criação de *tiles* responsivos e que podem ser dispostos em qualquer posição desejada, como pode-se ver na Figura 12.

Figura 12 – *Tiles* criados com Bulma.io.



Fonte: Bulma.io (2017)

3.2.6 Node.js e Express.js

Node.js é uma plataforma de desenvolvimento JavaScript com o objetivo de ser usada em aplicações em que a comunicação entre servidor e usuário precisa ocorrer em tempo real.

Sua arquitetura é conduzida por eventos assíncronos, projetado para criar aplicativos de rede escaláveis, utilizando um modelo de entrada e saída direcionada a evento não bloqueante tornando-se mais leve e eficiente, ideal para aplicações em tempo real com troca intensa de dados através de dispositivos distribuídos. (NODEJS, 2017)

O problema de algumas aplicações *web* é a quantidade de conexões concorrentes que o servidor pode manipular, com o Node.js não há esse problema pois ele não cria uma nova *thread* para cada requisição, ela apenas dispara um evento, garantindo que nunca vai ocorrer um *deadlock*.

O Express.js é um *framework web* estruturado, escrito em JavaScript que roda sobre o ambiente Node.js em tempo de execução. (MOZILLA, 2017a)

A filosofia do Express.js é fornecer ferramentas pequenas e robustas para servidores HTTP, tornando uma ótima solução para aplicativos de página única, sites, sistemas híbridos ou APIs HTTP públicas. (EXPRESSJS, 2017)

3.2.7 *Web Playback* SDK

O *Web Playback* SDK é uma biblioteca Javascript que pode ser integrada do lado do cliente. Permite criar um *player* do *Spotify* e reproduzir faixas de áudio através do navegador por meio de extensões de mídia criptografadas. (SPOTIFY, 2018)

Além disto, ele também permite a obtenção de dados da atual música em reprodução, dados sobre as músicas escutadas pelo usuário, e por fim, o controle sobre a reprodução local (*play*, *pause*, troca de músicas, entre outros).

Entretanto, a SDK não permite o *download* de áudio ou vídeo, autenticação do usuário do *Spotify* (para isso é utilizado a API) e a reprodução de uma certa faixa de áudio provinda de uma URL do *Spotify*.

3.2.8 Git

Git é um *software* de controle de versão para repositórios. Registra ao longo do tempo as mudanças nos arquivos de forma que as versões possam ser recuperadas futuramente. (CHACON, 2014). Atualmente o Git é muito utilizado por projetos de código aberto de diversas linguagens de programação.

É possível configurar o repositório local e sincronizá-lo com um repositório remoto

por meio de *push* (enviar mudanças) e *pull* (buscar mudanças). Promovendo a divisão do trabalho e removendo o problema de sincronização de código entre diferentes máquinas e usuários. (SCHMITZ, 2015)

Qualquer alteração feita no código pode ser visualizada por todos que estão trabalhando no projeto e as mesmas somente são incluídas no repositório remoto caso todos aprovem. Caso seja necessário voltar algumas versões por ocorrência de *bugs* no código, mal funcionamento de uma função recém implementada ou outras razões, as alterações estarão documentadas e é facilmente encontrada.

4 DESENVOLVIMENTO

4.1 Estudo e Definição das APIs

Este projeto consiste no desenvolvimento de um *dashboard* para controle de projetos e ambiente com o foco na área de computação, sendo assim, foram estudadas diversas APIs para comporem os módulos, as quais deveriam atender as necessidades dos usuários.

Tendo como base os principais *softwares* que são utilizados em locais de trabalho relacionados à tecnologia de informação e as principais características que poderiam ser melhoradas para gerar um melhor controle sobre o projeto e ambiente, as APIs escolhidas para conterem os módulos foram:

a) ***Github***: plataforma de hospedagem de código-fonte utilizado para controle de versões.

b) ***Slack***: plataforma de comunicação utilizado principalmente por empresas;

c) ***Facebook***: mídia social;

d) ***Instagram***: mídia social

e) ***Spotify***: serviço de *streaming* de música, *podcast* e vídeo.

f) ***SInteg***: projeto de integração de sensores.

Sendo que *Github* e *Slack* auxiliam no controle de projetos, *Facebook* e *Instagram* auxiliam na propagação e divulgação do projeto, e por fim, para controle de ambiente, o *Spotify* e *SInteg*.

4.1.1 *Github*

Como todas as empresas que lidam com desenvolvimento de aplicações utilizam alguma plataforma para controle de versões, o *Github* foi escolhido pois abrange 11 milhões de desenvolvedores e 36 milhões de visitantes diários (MIOZZO, 2015). Ele auxilia não apenas no controle de versões mas também pode informar qual desenvolvedor que está participando do projeto realizou certa mudança.

Quando certa pessoa finaliza uma tarefa, suas modificações precisam ser revisadas e aceitas pelos membros da equipe, para que isso seja possível basta criar um *pull request*, após a revisão de todos, o código desenvolvido poderá ser inserido com todo o restante do código e a equipe poderá atualizar seu ambiente pessoal com as mudanças.

Um dos principais problemas encontrados é a delonga para os membros da equipe revisarem as mudanças, e muitas vezes este fato ocorria pois os mesmos não consultavam o *Github* diariamente e o *pull request* passava despercebido.

Outra característica importante que o *Github* possui é a possibilidade da criação de *milestones*, ou seja, tarefas que precisam ser realizadas até uma data final. A visualização de informações de *milestones* é extremamente importante para manter o desenvolvimento do projeto eficiente e garantir a entrega no prazo correto. Muitas tarefas podem ser criadas mas não necessariamente incluídas em uma *milestone*, sendo assim, algumas possuem mais urgência do que outras.

Estes foram dois problemas mais ocorrentes e que podem ser resolvidos com a integração da API do *Github* com o *dashboard*.

4.1.2 *Slack*

O *Slack* é utilizado em diversas empresas, não somente na área de computação. Permite que os usuários entrem em grupos específicos e conversem dentro desses grupos em canais fechados que podem ser organizados de acordo com tarefas ou projetos específicos. Pode-se realizar a troca de arquivos, escrita de mensagens e permite a colaboração em tempo real. (AGUILHAR, 2017)

Em grupos com muitas pessoas, a quantidade de mensagens é grande e podem surgir assuntos paralelos à finalidade do mesmo, sendo assim, muitas mensagens importantes são perdidas.

Há também funcionários que preferem não utilizar o *Slack* pelos motivos citados acima, que podem causar distração e improdutividade.

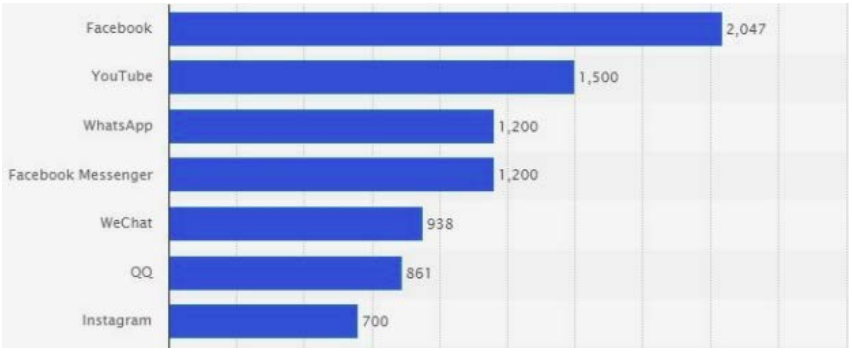
Quando uma mensagem importante é enviada, a utilização da *tag* @*here* é comumente utilizada, pois assim todos que estão utilizando o aplicativo recebem uma notificação sobre a mesma.

Para evitar problemas sobre a visualização de mensagens e também o excesso das mesmas, a integração da API do *Slack* pode ser utilizada e integrada com o *dashboard*. O mesmo pode apresentar as mensagens mais importantes, que foram enviadas com @*here* e também mensagens de um canal específico onde somente dados importantes são informados.

4.1.3 *Facebook e Instagram*

O *Facebook* e *Instagram* são duas das redes sociais mais populares no mundo, como pode ser visto na Figura 13, que apresenta a rede social e a quantidade de usuários em milhões.

Figura 13 – Redes sociais mais utilizadas no mundo.



Fonte: Kinast (2018)

Porém, estas duas redes sociais não são utilizadas somente para o entretenimento. Muitas empresas podem se beneficiar de outras maneiras, principalmente em relação ao marketing, garantindo que as marcas fortaleçam a sua imagem perante o seu público. (DRUBSCKY, 2018)

Para controlar o número de seguidores e as últimas menções de um perfil ou página pode-se utilizar a API do *Facebook* e *Instagram* e integrá-las com o *dashboard* para garantir uma melhor visualização da eficiência da campanha de marketing de uma empresa.

4.1.4 *Spotify*

O *Spotify* é um serviço de *streaming* de áudio, pode ser acessado pela *web*, aplicativos instalados no computador ou até mesmo através de um dispositivo móvel. (SPOTIFY, 2018b)

O aplicativo permite a criação de *playlists* colaborativas, ou seja, as pessoas autorizadas podem adicionar, excluir e reordenar as faixas. Característica interessante quando deseja-se executar as faixas de áudio em um ambiente compartilhado.

Para ambientes de trabalho mais descontraídos ou salas de esperas, a utilização do *Spotify* pode deixar o ambiente mais agradável. A integração da API deste *software* permite o controle sobre uma determinada *playlist* e também a visualização de dados sobre a música em reprodução.

4.1.5 *SInteg*

O *SInteg* é um projeto de conclusão de curso desenvolvido por Matheus Coelho Solha que engloba quatro sensores, sendo estes de temperatura, umidade, luminosidade, também conhecido como LDR, e o sensor magnético *reed switch*.

Os sensores são instalados em um ambiente e através de rotinas previamente desenvolvidas, são disponibilizadas as informações captadas pelos sensores.

O sensor LDR pode determinar se a lâmpada do local em que ele foi instalado está acessa ou não, informação útil quando uma empresa possui uma grande quantidade de cômodos ou também quando o expediente for finalizado e deseja-se verificar se todas as lâmpadas foram desligadas.

O sensor *reed switch* é um sensor magnético composto por duas lâminas de ferro. Normalmente instalado em portas para determinar se estão abertas ou fechadas, informação conveniente quando os sensores são instalados em salas de reuniões, onde podem determinar se as mesmas estão ocupadas ou não.

A integração do *SInteg* com o *dashboard* pode facilitar a visualização de informações do ambiente de trabalho e garantir uma rápida tomada de decisões de acordo com os dados disponibilizados.

4.2 Desenvolvimento do Servidor

Para obter os dados de cada API foi desenvolvido um servidor utilizando Node.js, uma vez que sua arquitetura é conduzida por eventos assíncronos e seu modelo de entrada e saída é não bloqueante, característica importante para uma aplicação onde diversas requisições são realizadas ao mesmo tempo.

Juntamente com o Node.js, foi escolhido o uso do *framework* Express.js, o qual cria abstrações de rotas, permite requisições de diferente tipos (*POST, GET, DELETE*), a configuração de portas para conexão e para a resposta da mesma e também permite a adição de *middlewares*. (MOZILLA, 2017b)

Primeiramente, para testes, todas as requisições, sendo elas de autorização e de obtenção de informações, foram realizadas no servidor.

4.2.1 Instalação e Configuração do Node.js e Express.js

Para conseguir utilizar o Node.js e realizar requisições para as APIs é necessário instalar o gerenciador de pacotes Node (NPM) em sua máquina. Para isto, basta realizar o download da versão mais nova do Node.js de acordo com o seu sistema operacional utilizado, será instalado tanto o Node.js quanto o NPM.

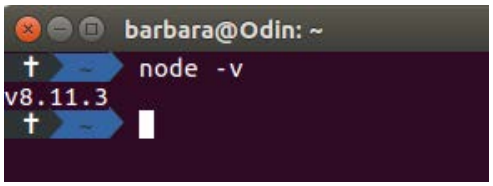
Todas as versões podem ser encontradas no site oficial do Node.js. [1]

Caso seja necessário verificar se a instalação ocorreu com sucesso e qual versão foi instalada, pode-se executar o comando mostrado na Figura 14. Para instalação do

[1] <https://nodejs.org/en/>

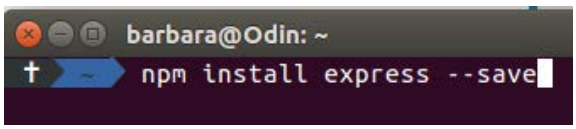
Express.js, basta executar o comando mostrado na Figura 15.

Figura 14 – Verificação da Versão do Node.js.



Fonte: elaborado pelo autor

Figura 15 – Instalação do Express.js.



Fonte: elaborado pelo autor

Para criar o servidor em Express.js e configurá-lo a fim de que o mesmo receba respostas em uma porta específica basta seguir o mesmo código da Figura 16. Neste exemplo, sempre que abrir o navegador na porta especificada, aparecerá a mensagem "*Dashboard Miser*!".

Figura 16 – Começando um Servidor com Node.js e Express.js.

```
const express = require('express');
const app = express();

app.get('/', function (req, res) {
    res.send('<h1>Dashboard Miser!</h1>');
});

app.listen('4392', function () {
    console.log('listening on 4392');
});
```

Fonte: elaborado pelo autor

4.2.2 Cadastro das aplicações

Para possibilitar a integração e obtenção de dados da API do *Slack*, *Spotify* e *Github* com o servidor local deste projeto, foi necessário cadastrar as contas em seus respectivos sites para desenvolvedores [2] [3], para o *Slack* o nome dado à esta função é *App* e para o *Spotify* é *Dashboard*.

Para o *Github* basta entrar na sua conta, criar um *GitHub App* através da opção *Developer Settings* e associá-lo com o repositório que compartilhará as informações.

Com o cadastro realizado pode-se obter os dados necessários para realizar futuras requisições, como o *client id* e o *client secret*, e também é possível editar informações como

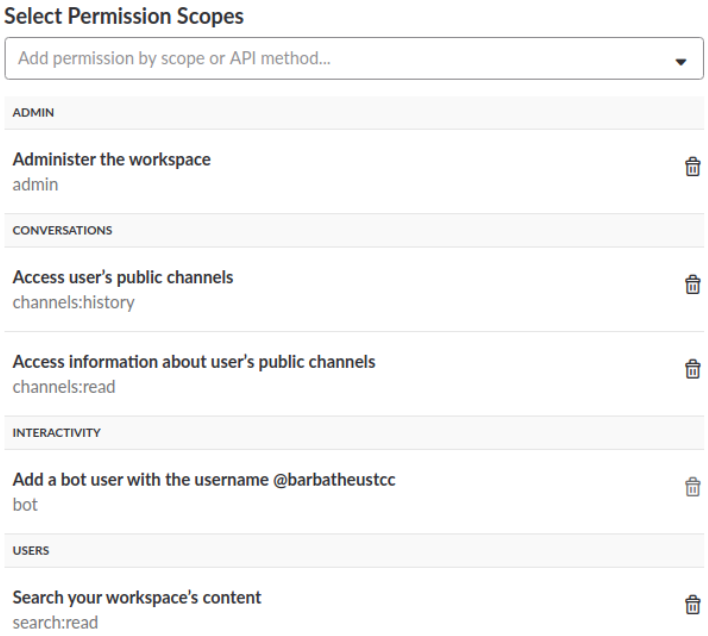
[2] https://api.slack.com/apps
[3] https://developer.spotify.com/dashboard/login

a URL de redirecionamento, ou seja, a URL do servidor local para onde os dados serão retornados.

Para o *App* do *Slack*, é necessário informar o tipo de dado que seu servidor deseja acessar, pois somente assim as permissões serão confirmadas. Estes tipos de dados são chamados de escopo, e sua explicitação durante a requisição realizada pelo servidor também é necessária.

Caso um tipo de escopo desejado não tenha sido previamente adicionado ao *App* e informado durante a requisição do servidor, mesmo que o usuário autorize a obtenção dos dados, a API não providenciará as informações. A Figura 17 apresenta os escopos que foram adicionados ao *App* utilizado neste projeto.

Figura 17 – Escopos permitidos pelo *App* do *Slack*.



Fonte: Slack (2018a)

Cada escopo do *Slack* é referente a alguma informação específica. O escopo *channels:history*, por exemplo, foi inserido na lista de permissões pois o mesmo é utilizado para obter mensagens e eventos de um canal. Para verificar qual escopo é necessário para determinada função, basta olhar na documentação[4].

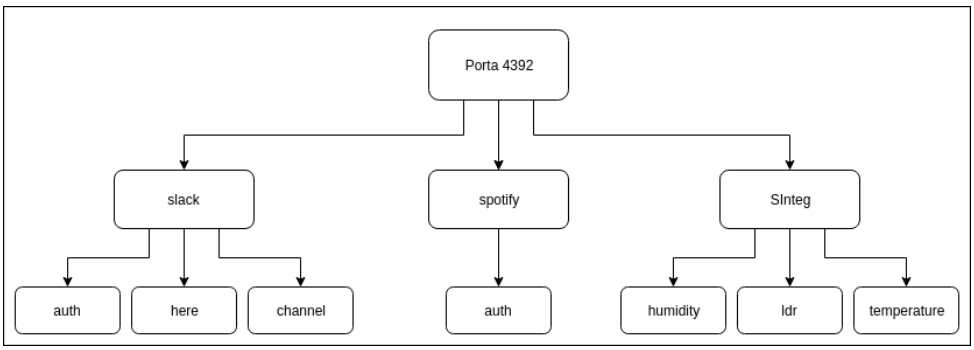
Para o *Spotify* não é necessário informar os escopos desejados no *Dashboard*, sua explicitação durante a requisição já basta. O único dado de informe necessário é a URL de redirecionamento, visto que sem a mesma, a API não retorna os dados por questões de segurança.

[4] https://api.slack.com/methods

4.2.3 Requisições de Autorização

Como mostrado anteriormente na Figura 16, a porta escolhida para receber e transmitir dados do servidor é a 4392, a partir da mesma foram criados diversos caminhos para realizarem requisições específicas, sendo estas para o *Spotify*, *Slack* e *SInteg*, como mostra a Figura 18.

Figura 18 – Caminhos do servidor a partir da porta pré definida.



Fonte: elaborado pelo autor

Tendo configurado o Node.js, cadastrado os aplicativos e definido suas rotas, foi necessário realizar uma requisição de autenticação para a API do *Slack* e *Spotify*.

O respositório do *Github* sincronizado com o servidor deste projeto é público, sendo assim, as informações podem ser acessadas sem a obtenção de autorização do dono. Consequentemente, esta etapa de autorização não foi utilizada para o *Github*.

Para realizar uma requisição é necessário informar a URL para qual está sendo feita, que neste caso são as URLs de autenticação[5,6] de cada API. Para as duas, é necessário complementar a requisição com o *client id* que foi obtido no cadastro e o escopo, que são as informações que deseja-se obter. Para o *Spotify* também é preciso informar a URL de redirecionamento, que deve ser a mesma cadastrada no *Dashboard*.

Como esta primeira requisição é utilizada somente para solicitar uma URL, que no caso é a de autenticação, é utilizado o método GET.

Com a realização desta requisição, o usuário é redirecionado à página de autenticação de cada API onde poderá escolher entre autorizar ou negar o acesso aos dados requeridos pelo servidor. Caso a autorização seja confirmada, um *authorization token* é retornado ao servidor na URL de redirecionamento.

A segunda etapa na autorização é outra requisição, que neste caso utiliza o método POST, pois as informações enviadas no corpo da requisição são utilizadas para criar um

[5] https://accounts.spotify.com/authorize
[6] https://slack.com/oauth/authorize

novo recurso, que será um *access token*.

Para dar continuidade à autorização, a requisição precisa conter a nova URL para a qual será feita[7,8], o *authorization code* que foi retornado anteriormente, o tipo de *code* que está sendo mandado, que neste caso é o *authorization code*, o *client id* e o *client secret* que foram obtidos no cadastro. A Figura 19 apresenta esta requisição feita para o *Slack*.

Caso todos os dados enviados estejam corretos, a API retorna um *access token* para o servidor, este *token* é utilizado em todas as futuras requisições, pois somente com ele os dados requisitados são informados ao servidor.

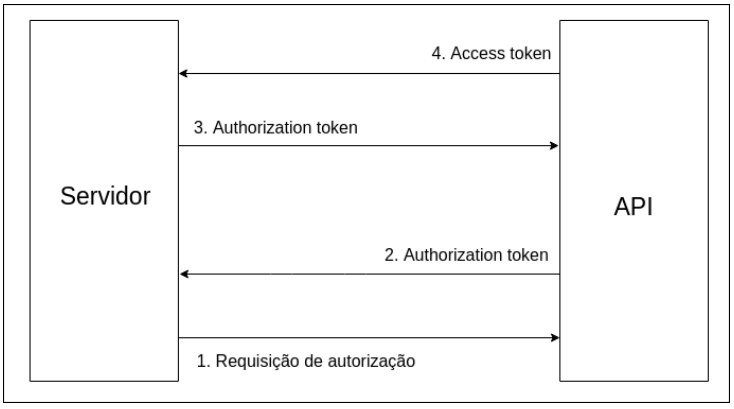
Figura 19 – Requisição para obter o *access token Slack*.

```
app.get('/slack/token', (req, res) => {
    let data = querystring.stringify({
        client_id: encodeURI(process.env.SLACK_CLIENT_ID),
        client_secret: encodeURI(process.env.SLACK_CLIENT_SECRET),
        code: encodeURI(req.query.code),
    });

    axios.post(process.env.SLACK_AUTH_ACCESS, data, {
        headers: {
            'Content-type': 'application/x-www-form-urlencoded',
        },
    })
}
```

Fonte: elaborado pelo autor

O fluxo de requisições e informações trocadas entre o servidor e a API é apresentado na Figura 20.

Figura 20 – Fluxo de requisições entre o servidor e APIs.



Fonte: elaborado pelo autor

[7] https://slack.com/api/oauth.access
[8] https://accounts.spotify.com/api/token

O *access token* e o *client secret* são dados que devem ser mantidos seguros e não podem ser revelados à terceiros. Para armazenamento dos mesmos, foi utilizado um módulo NPM chamado *dotenv*. Ele foi desenvolvido para armazenar informações sensíveis sem que as mesmas fiquem expostas a usuários ou *softwares* de controle de versão, como o *Github*. (NPM, 2018)

O *dotenv* é um arquivo que deve ser criado na raiz do diretório e nele pode-se armazenar todos os dados desejados, sejam eles sensíveis ou não. Para utilização das variáveis em algum outro arquivo do projeto, basta inserir no código 'process.env' e o nome da variável que foi salvo no arquivo. A sua utilização pode ser vista na Figura 19, onde o *client id*, *client secret* e a URL de autorização foram armazenadas no arquivo *dotenv*.

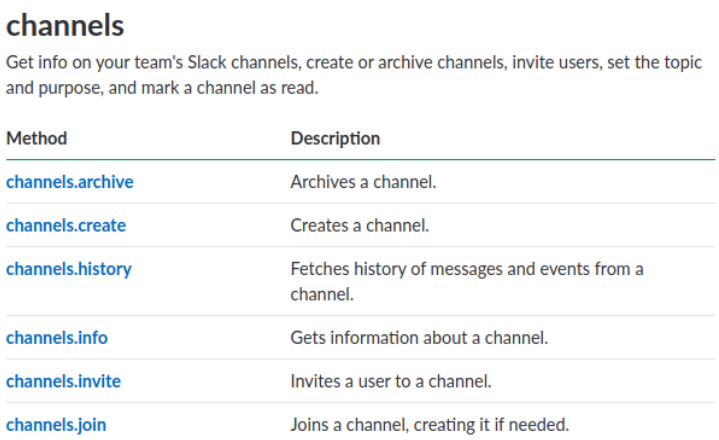
4.2.4 Requisições para obtenção de dados

Após a obtenção do *access token* pode-se realizar requisições para obtenção de dados do usuário.

Como o *Spotify* utiliza a sua SDK para reprodução das músicas, não é necessário realizar nenhuma requisição para a API além da autorização.

Para o *Slack*, devem ser realizadas diferentes requisições para cada informação desejada. Em sua documentação pode-se encontrar vários métodos, que são divididos em canais, chat, conversas, grupos, e diversos outros. Cada método possui uma finalidade e retorna diferentes dados para o servidor. Na Figura 21 pode-se visualizar alguns métodos específicos para canais.

Figura 21 – Métodos específicos para canais do *Slack*.



channels

Get info on your team's Slack channels, create or archive channels, invite users, set the topic and purpose, and mark a channel as read.

Method	Description
channels.archive	Archives a channel.
channels.create	Creates a channel.
channels.history	Fetches history of messages and events from a channel.
channels.info	Gets information about a channel.
channels.invite	Invites a user to a channel.
channels.join	Joins a channel, creating it if needed.

Fonte: Slack (2018b)

No desenvolvimento das requisições para a obtenção de dados do *Slack* foram utilizados três métodos, sendo estes o *search.messages*, *conversations.list* e *channels.history*.

O *search.messages* foi utilizado para obter as mensagens com as *tags* *@here* encontradas nos canais da conta vinculado ao servidor. A requisição para obtenção destes dados pode ser vista na Figura 22.

Figura 22 – Requisição para obter dados do *Slack*.

```
app.get('/here', (req, res) => {
    var str = process.env.SLACK_BASE + process.env.SLACK_SEARCH_MSG
    + '?token=' + slackToken
    + '&count=3'
    + '&query=@here'

    axios.get(
        encodeURI(str)
    ).then(result => {
        var response = {
            channel: result.data.messages.matches[0].channel.name,
            message: result.data.messages.matches[0].text,
        };
        res.json(response);
    });
})
```

Fonte: elaborado pelo autor

Para a requisição utilizando este método é necessário informar o *access token* e o *query*, que é a palavra ou conjunto a ser procurado nos canais. Algumas informações podem ser adicionadas para filtrar os resultados, como por exemplo o *count*, que informa a quantidade de resultados a serem retornados.

Como a resposta para o servidor exibe diversas informações, para esta requisição foram guardados apenas o texto da mensagem, usuário que a enviou e canal para qual foi enviada.

O método *conversations.list* lista todos os canais criados e suas informações, esta requisição foi apenas realizada para obter o *id* dos canais para que fosse possível utilizar o método *channels.history*.

O *channels.history* foi utilizado para obter as mensagens e eventos de um canal específico. Para realizar requisição utilizando este método é necessário informar o *access token* e o *id* do channel que se deseja obter as informações. Os dados salvos para futura exibição no *dashboard* foram as mensagens do devido canal e o usuário que as enviou.

As informações requisitadas a API do *Github* foram sobre projeto, repositório, *pull requests* e *milestones*.

Para realizar uma requisição basta informar a URL, que varia de acordo com a informação desejada, o *client id* e o *client secret*, que foram obtidos ao cadastrar o repositório.

A fim de se obter informações sobre o projeto foi feita a requisição apresentada na Figura 23. Na URL é descrito o nome do repositório (mabacs) e o nome do projeto (*Dashboard*), acoplado com o *client id* e o *client secret*. Os dados armazenados provindos

da resposta da requisição foram o nome do projeto, a quantidade de *issues* abertas, a quantidade de usuários que se subscreveram no projeto e a quantidade de *pull requests* abertos.

Figura 23 – Requisição para obter dados sobre um projeto do *Github*.

```
axios.get('https://api.github.com/repos/mabacs/Dashboard'
    + this.client_id
    + this.client_secret
    ).then(res => {
        this.projectName = res.data.name;
        this.projectIssues = res.data.open_issues_count;
        this.projectStarts = res.data.stargazers_count;
        this.projectSubscribers = res.data.subscribers_count;
    });
```

Fonte: elaborado pelo autor

Também foi realizada uma requisição para obter informações do repositório, como a imagem que foi definida para o mesmo.

E por fim, foi realizada uma requisição para obter dados sobre as *milestones*, sobre a mesma foram salvos o título, a quantidade de *issues* abertas e finalizadas, a descrição da mesma e a data final de entrega.

Para cada requisição a URL é modificada, para visualização dos dados necessário basta seguir a documentação que se encontra no site oficial[9] da API do *Github*.

As informações sobre o *SInteg* foram obtidas através de requisições feitas para um outro servidor, não foi necessário realizar autenticação, apenas a requisição direta ao servidor, como apresentado na Figura 24. Na mesma foi explicitado a URL para requisição e foi retornado o valor do sensor, que seria *on* ou *off*.

Figura 24 – Requisição para obter dados provindos do sensor de luminosidade do *SInteg*.

```
app.get("/sinteg/ldr", (req, res) => {
    var request = {
        method: 'get',
        url: 'http://raspdock.duckdns.org:8123/api/states/sensor.ldr'
    };

    axios(request).then(resAxios => {
        res.send(resAxios.data);
    }).catch(error => {
        res.send(error.message);
    });
});
```

Fonte: elaborado pelo autor

[9] https://developer.github.com/v3/

Para obter os outros dados sobre os três sensores foram realizadas outras três requisições semelhantes à citada acima, apenas modificando o final da URL, trocando 'ldr' por *temperature*, *humidity* e *reed_switch*.

4.3 Desenvolvimento dos módulos

Cada API escolhida é um módulo do *dashboard* e para melhor visualização dos dados foi estudado o *framework* Bulma.io, o qual é totalmente responsivo e modular.

Sendo assim, as informações de cada módulo foram agregadas em um *tile*, sendo este um *layout* que auxilia no *design* da estrutura da página.

Para cada elemento do *dashboard* foi decidido um estilo de cores e *layout*, que foi definido em um arquivo Sass, que é integrado com o Bulma.io assim que o sistema é iniciado na *web*. Como o Bulma.io já possui cores e estilos pré definidos para cada componente, também foi necessário alterá-los para que houvesse harmonia entre todos os elementos do *dashboard*.

A definição das cores e estilos em apenas um arquivo facilita no controle do projeto, pois caso seja necessário realizar alguma alteração, basta mudar em apenas um arquivo. Um exemplo do conteúdo do arquivo pode ser visto na Figura 25.

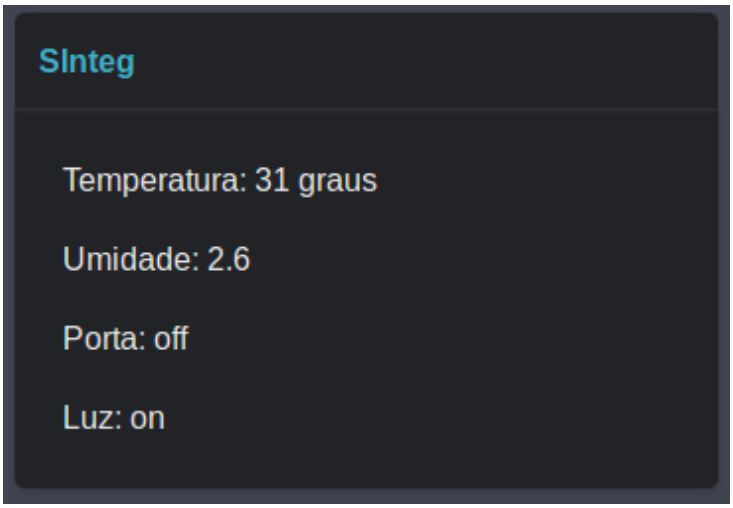
Figura 25 – Definição de cores e estilos.

```
You, a few seconds ago | 1 author (You)
@charset "utf-8";
// bulma card variables
$card-color: #e8e8e8;
$card-background-color: #222326;
$card-header-color: #e8e8e8;
$card-header-shadow: 0 1px 2px rgba(#fff, 0.1);
$card-header-color: #4bdcffc4;

$border-radius: 5px;
$background-grey: #404450;
$card-grey: #222326;

@import "~bulma/bulma";
```

Fonte: elaborado pelo autor

Tendo definido que para o layout da página seriam utilizados os *tiles*, foi decidido que dentro de cada um teria um *card*, para dispor as informações de uma maneira mais agradável para o usuário. Um exemplo se encontra na Figura 26 (um protótipo do módulo do *SInteg*), o mesmo é um *tile* e inserido nele há um *card*.

Figura 26 – Protótipo do módulo *SInteg*.



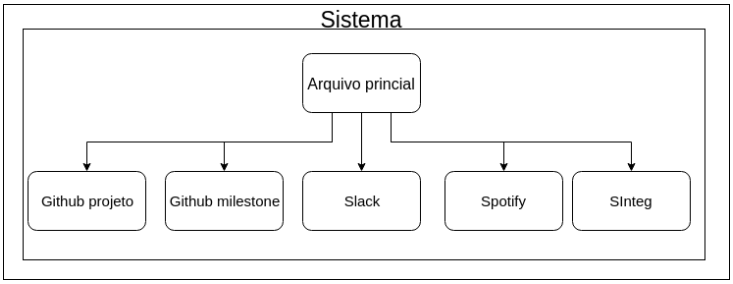
Fonte: elaborado pelo autor

Um *card* possui título e o corpo. Dentro do título pode-se colocar tanto texto quanto imagens. No desenvolvimento do *card* do *Github* foi inserido o título de acordo com o nome do projeto e a imagem que foi encontrada no repositório do mesmo.

A personalização de cada *card* é possível devido a utilização de classes disponíveis pelo Bulma.io e também pela utilização de propriedades definidas no CSS e SASS.

A utilização do Vue.js permitiu que os módulos fossem desenvolvidos separadamente, sendo assim, cada módulo possui suas próprias características em javascript. A junção dos módulos foi realizada no arquivo principal. A Figura 27 representa a estrutura do sistema.

Figura 27 – Estrutura dos componentes do sistema.



Fonte: elaborado pelo autor

4.4 Integração do Servidor com o *Dashboard*

Para obter os dados específicos é necessário utilizar APIs, porém, os dados precisam ser protegidos de pessoas e programas maliciosos. Sendo assim, só é possível obter informações de uma conta ou usuário específico se houver a autorização do mesmo.

O processo de autorização deve ser realizado pelo servidor, pois como resposta destas requisições é recebido um *access token*. Este *token* deve ser armazenado no servidor e somente disponibilizado para o *frontend* quando for requisitado. Entretanto, o *token* precisa permanecer fora do alcance de usuários.

Primeiramente, todas as requisições foram realizadas no servidor. Com a integração do *dashboard* com o *servidor*, as requisições de dados foram movidas para o *frontend*, deixando apenas as requisições de autorização no *backend*.

Porém, as requisições de dados também necessitam do *access token* para serem executadas com sucesso. Este dado foi transmitido ao *frontend* através de *cookies* HTTP.

Os *cookies* são utilizados para enviar informações do servidor para o navegador. É utilizado para realização de *logins* e armazenamento de informações. (CISCO, 2018)

Quando o usuário iniciar o *dashboard*, poderá conectar com a sua conta nos módulos do *Slack* e do *Spotify* para que seja possível obter as informações.

Para que o usuário consiga autorizar a coleta de dados, uma requisição é feita do *frontend* para a URL de autorização de cada API, a qual instantaneamente disponibiliza uma tela de autorização para o usuário, onde o mesmo tem a opção de aceitar ou rejeitar a autorização, e caso não esteja conectado a uma conta ainda, possibilitará a realização do *login*. A Figura 28 apresenta a tela de autorização para qual o usuário foi redirecionado pela API do *Slack* ao clicar no botão para se conectar.

Após a autorização do usuário, a API redireciona a sua resposta contendo o *authorization token* para o servidor, o qual realiza outra requisição para a API.

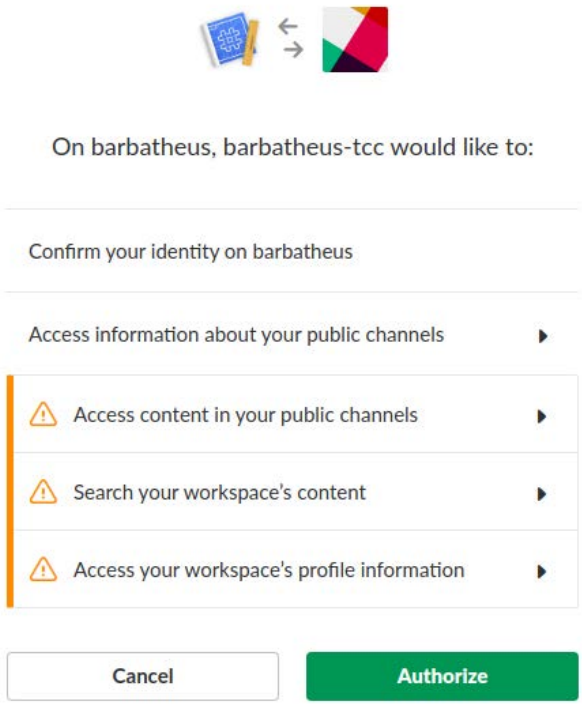
Esta requisição retorna o *access token* para o servidor. Por fim, o servidor retorna para o *frontend* um *cookie* contendo o *access token* para que o mesmo possa ser utilizado em novas requisições de obtenção da informações.

A Figura 29 mostra o fluxo de dados e requisições entre a API, o servidor e o *fronted*.

Este fluxo entre o *frontend*, *backend* e a API garante a integridade e segurança dos dados.

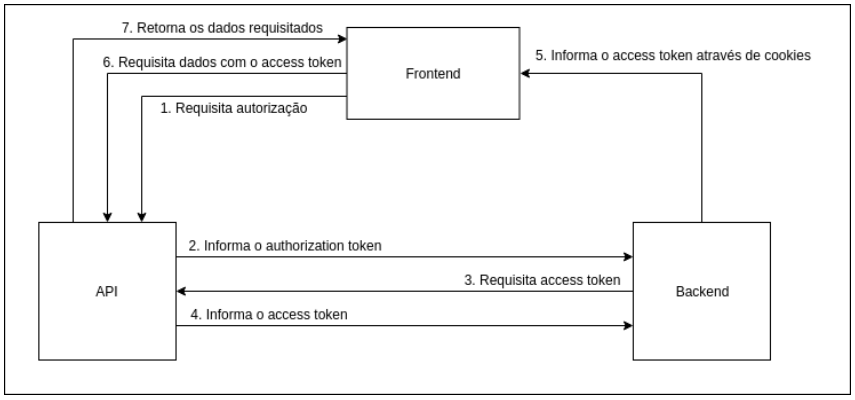
Em cada módulo foi realizada a integração com o servidor da maneira descrita acima, para que fosse possível a realização da autorização provinda do usuário e a obtenção do *access token* salvo no servidor. Cada requisição de dados foi movida para o módulo de seu respectivo componente.

Figura 28 – Redirecionamento para página de autorização do *Slack*.



Fonte: elaborado pelo autor

Figura 29 – Fluxo de dados e requisições entre API, servidor e *frontend*.



Fonte: elaborado pelo autor

As Figuras 30 e 31 apresentam a integração do *frontend* com o *backend* e as APIs. Primeiramente o usuário é redirecionado pela API para realizar a autorização, posteriormente é obtido o cookie disponível na página e por fim são chamados os métodos responsáveis pela realização das requisições que obtêm os dados que serão dispostos no módulo.

Figura 30 – Requisição de autorização para o usuário.

```
slackAuthRequest() {
    const Authpath = 'https://slack.com/oauth/authorize'
    + '?client_id=417514605056.418759063329'
    + '&scope=search:read '
    + 'channels:read '
    + 'channels:history '
    + 'users:read';

    const winClosed = setInterval(() => {
        if (win.closed) {
            clearInterval(winClosed);
            this.slackGetMessages();
            this.slackGetReunioes();
            this.slackGetDevops();
        }
    }, 100);
},
```

Fonte: elaborado pelo autor

Figura 31 – Requisição do servidor para a API.

```
app.get('/slack/token', (req, res) => {
    let data = querystring.stringify({
        client_id: encodeURI(process.env.SLACK_CLIENT_ID),
        client_secret: encodeURI(process.env.SLACK_CLIENT_SECRET),
        code: encodeURI(req.query.code),
    });

    axios.post(process.env.SLACK_AUTH_ACCESS, data, {
        headers: {
            'Content-type': 'application/x-www-form-urlencoded',
        },
    }).then(resAxios => {
        res.cookie('slackToken', token);
        res.end()
    });
});
```

Fonte: elaborado pelo autor

Tendo realizado a integração do servidor com o *frontend* para a captura dos dados que seriam expostos, foi possível realizar o desenvolvimento de cada módulo.

4.4.1 *Github*

O *Github* é muito utilizado na área de computação pois permite o controle de versão, registrando as mudanças no código e permitindo a organização no trabalho em grupo.

Para este módulo foi decidido a construção de dois *cards*, sendo que o primeiro apresenta uma visão geral do repositório e no segundo é possível visualizar informações sobre uma *milestone*.

No primeiro *card*, visto na Figura 32, o título se refere ao nome do projeto, seguido da imagem que se encontra em seu repositório. As informações disponíveis são:

a) **Estrelas**: quando um usuário seleciona a estrela em algum projeto, o mesmo torna-se mais fácil de ser encontrado e o *Github* também pode sugerir para o usuário algum outro projeto que seja similar;

b) ***Subscribers***: quando um usuário se inscreve em um projeto, permite que receba notificações sobre novas discussões, bem como eventos em seu *feed* de atividades;

c) ***Issues* abertas**: a quantidade de *issues* abertas no projeto e que ainda não foram finalizadas;

d) ***Pull requests***: quando alguma *issue* é finalizada, pode-se criar um *pull request* para que os membros do projeto vejam as modificações, permitindo também a sugestão de alterações. Quando um *pull request* é aceito por todos os membros que foram requisitados, o código poderá ser inserido no *branch* em que se encontra todo o código.

No segundo *card*, mostrado na Figura 33, se encontra o título da *milestone* seguido da imagem que se encontra em seu repositório. As informações disponíveis são:

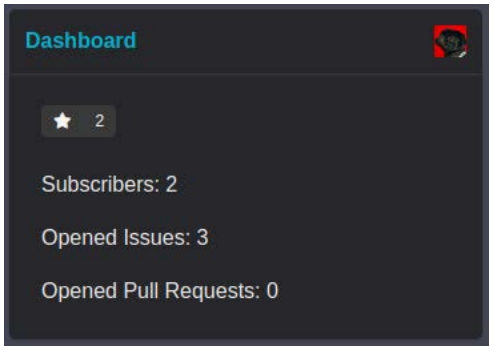
a) **Descrição**: a descrição da *milestone*;

b) **Data final**: data final para entrega de todas as *issues* que estão inseridas na *milestone*;

c) ***Issues* abertas**: a quantidade de *issues* dentro da *milestone* que ainda não foram finalizadas.

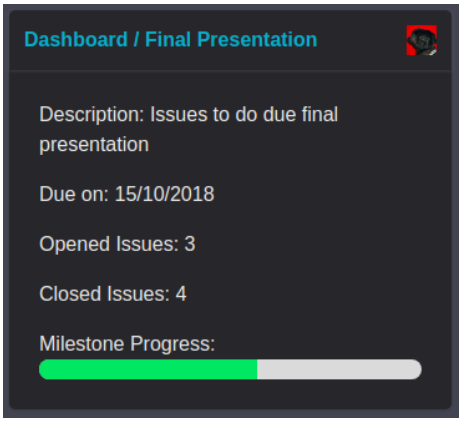
d) **Progresso**: barra de progresso para indicar a quantidade de *issues* finalizadas e abertas.

Figura 32 – *Card* com as informações do projeto do *Github*.



Fonte: elaborado pelo autor

Figura 33 – *Card* com as informações da milestone de um projeto do *Github*.



Fonte: elaborado pelo autor

Para obtenção destes dados foram realizadas três requisições direcionadas para o repositório, o projeto e para os *pull requests* do projeto.

As informações apresentadas neste módulo são provindas do repositório deste projeto. Na criação do repositório foi selecionado a opção do mesmo ser público, sendo assim, não foi preciso realizar requisições de autenticação e consequentemente não foi necessário obter o *access token*.

4.4.2 *Slack*

O *Slack* é uma das principais formas de comunicação em empresas por conseguir abranger todos os funcionários, permitir a criação de grupos, lembretes e também avisos.

O canal em que todos os membros da equipe se encontra é denonimado *general* e é normalmente neste canal que os lembretes ou mensagens importantes são mandados. Quando deseja-se que todos recebam notificação sobre uma certa mensagem, basta inserir

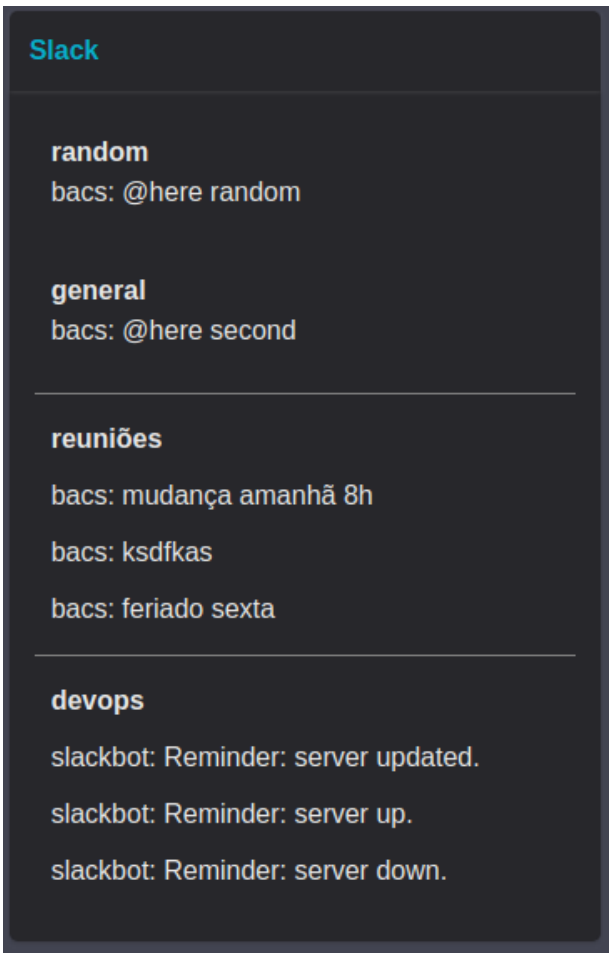
o comando *@here* na mesma, sendo assim, todos são alertados e podem responder a mensagem o mais rápido possível.

Para este módulo foi decidido captar as mensagens com a *tag* *@here*, pois estas são as mensagens mais importantes e que a finalidade é atingir a todos.

Também, para este módulo, serão mostradas as três últimas mensagens de dois canais específicos, que neste caso foram escolhidos os canais "reuniões"e "*devops*", que mostra mensagens sobre servidores e infraestrutura.

A Figura 34 apresenta o *card* do *Slack* com todas as informações obtidas pelas requisições.

Figura 34 – *Card* com as informações do *Slack*.



Fonte: elaborado pelo autor

Para obter as informações do *Slack* o usuário precisa se conectar à sua conta. Primeiramente é pedido que o usuário se conecte em sua conta.

Para o desenvolvimento do login, é utilizada a diretiva *v-click* do Vue.js, a qual é acoplada a um botão e quando o mesmo é pressionado, um método é executado.

Neste caso, assim que o botão é acionado, é realizada a requisição para a URL de autorização do *Slack*, a qual abre uma nova guia no navegador para que o usuário possa realizar o *login* e autorizar a coleta de dados. Após a autorização do usuário, é retornado ao servidor o *access token* e o mesmo é retornado para o *frontend* através de um *cookie*.

Quando um *cookie* é criado, é necessário associá-lo a um nome, no caso do *Slack* foi escolhido o nome *slackToken*.

Para que o *frontend* consiga captar o *token* contido no *cookie*, uma função é executada assim que a página de autorização é fechada.

Após a obtenção do *access token* são realizadas outras três requisições pelo *frontend*. Sendo a primeira para acessar as mensagens que contém a *tag @here* no canal *general*, a segunda para acessar as mensagens do canal reuniões, e a última para obter as mensagens do canal *devops*.

O *Slack* relaciona o nome dos usuários com seus respectivos IDs, quando as requisições retornam uma resposta, a mesma contém o ID do usuário, e não o seu nome.

Para melhor entendimento ao se observar o dashboard, foi realizada outra requisição, sendo esta para obter o nome do usuário através do ID retornado nas requisições anteriores. A Figura 35 apresenta a função que realiza a requisição que retorna o nome dos usuários dado o seu ID.

Figura 35 – Função que obtém o nome do usuário a partir de seu ID no *Slack*.

```
async getUser(userId) {
    const pathUser = `${'https://slack.com/api/users.info'
    + '?token='}${this.getCookie('slackToken')
    }&user=${userId}`;

    const { data } = await axios.get(pathUser);

    return data.user.profile.display_name;
},
```

Fonte: elaborado pelo autor

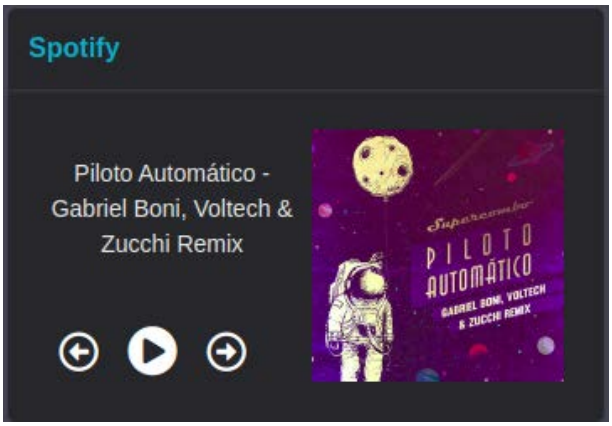
4.4.3 *Spotify*

Em ambientes mais descontraídos as empresas podem utilizar o *Spotify* para manter o local mais agradável, como por exemplo, em salas de espera ou até mesmo locais de trabalho com menos funcionários.

As requisições realizadas na API do *Spotify* permitem a visualização de dados, porém o controle sobre a uma *playlist* só pode ser realizada com o *Web Playback* SDK.

Como visto na Figura 36, este módulo permite o controle de uma *playlist* e também mostra informações, como o nome da atual música em reprodução e a capa do álbum. Para o controle do módulo, basta conectar com uma conta do *Spotify*.

Figura 36 – *Card* com as informações do *Spotify*.



Fonte: elaborado pelo autor

O desenvolvimento deste módulo foi similar ao módulo do *Slack*. O usuário precisa conectar em sua conta através de um botão que quando acionado realiza uma requisição para a API do *Spotify*. O usuário é redirecionado para uma nova página que contém a opção de *login* que após ser realizado retorna um *access token* para o servidor.

Assim que o usuário acessa sua conta, a função responsável pela conexão com a SDK é chamada. Para utilizar o *player* é necessário adicioná-lo ao HTML da página, já que a SDK é o elemento que possibilita a execução de áudio na página *web*.

Para inserir a SDK na página HTML é utilizado um script que integra o *player* assim que o usuário conecta em sua conta, como apresentado na Figura 37.

Figura 37 – *Script* que insere o *player* do *Spotify*.

```
const myScript = document.createElement('script');
myScript.setAttribute('src', 'https://sdk.scdn.co/spotify-player.js');
document.head.appendChild(myScript);
```

Fonte: elaborado pelo autor

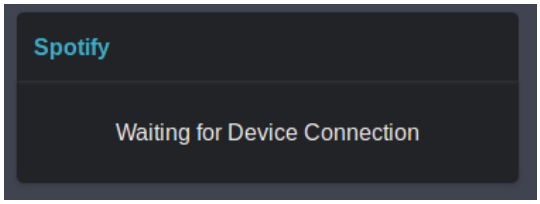
Após o *login* do usuário e a inserção da SDK no HTML o *player* pode ser inicializado. Para isto é necessário utilizar o *access token* que foi mandado pelo servidor através de um *cookie*.

A função que obtém o *cookie* da página é chamada e é retornado o *cookie* com o nome de *spotifyToken*, o qual foi determinado pelo servidor quando foi enviado.

Após a obtenção do token o *player* é inicializado e uma mensagem aparecerá para o usuário para que ele abra o *Spotify* e selecione a opção de reproduzir o áudio na *web*.

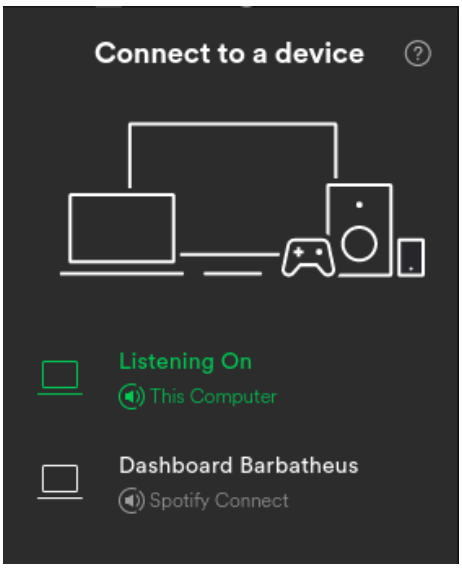
A Figura 38 mostra a mensagem de espera e a Figura 39 apresenta a seleção de dispositivos. Quando a opção de reprodução no dispositivo *Dashboard Barbatheus* for selecionada, o áudio passa ser reproduzido na página do *dashboard*.

Figura 38 – Mensagem para o usuário trocar o dispositivo de áudio.



Fonte: elaborado pelo autor

Figura 39 – Seleção de dispositivos do *Spotify*.



Fonte: elaborado pelo autor

A utilização da SDK permite o acoplamento de eventos que captam mudanças no estado do *player*. Sendo assim, sempre que houver a troca de música, outra função é chamada pra que esta obtenha os dados do novo áudio em reprodução e possa atualizar os dados dispostos no *dashboard*.

4.4.4 *SInteg*

O *SInteg* é um projeto de conclusão de curso de ciência da computação desenvolvido por Matheus Coelho Solha. Neste, foram integrados quatro sensores, os quais são de umidade, temperatura, luminosidade e *reed switch*, sendo que este último é um sensor magnético composto por duas lâminas de ferro.

O sensor de luminosidade, também conhecido como sensor LDR, pode determinar se a luz de alguma sala está ligada ou desligada. Mostra sua utilidade quando a empresa está finalizando o horário de atendimento e por alguma eventualidade houver o esquecimento de alguma lâmpada ligada, ou também quando alguma sala não está mais em uso e foi deixado alguma lâmpada ligada. Sendo assim, basta olhar no *dashboard* e observar os dados do módulo *SInteg*.

Visto que o sensor *reed switch* é magnético e a sua resposta varia com o encontro ou desencontro do mesmo, ele pode ser instalado em portas para revelar se a mesma está aberta ou fechada.

Sendo assim, a utilização deste sensor é vantajosa caso seja acoplado à portas de salas de reuniões. Caso um sala de reunião seja distante da sala de trabalho, basta olhar nas informações do módulo para verificar se a sala está ou não em uso.

Para obter as informações foram realizadas quatro requisições para o servidor do *SInteg*. As requisições são feitas para o mesmo domínio e porta, a diferença entre cada uma é o caminho. A Figura 40 apresenta as URLs para quais foram feitas as requisições.

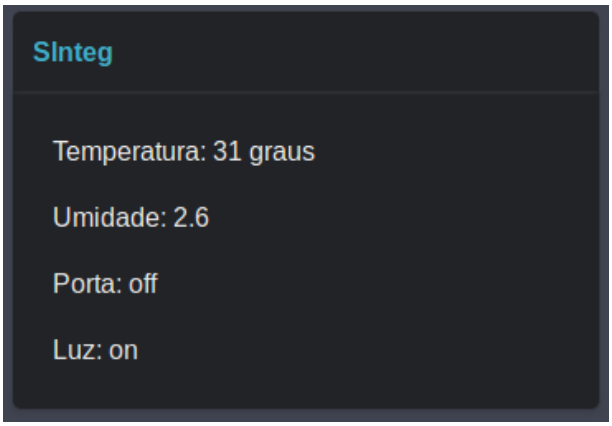
Figura 40 – URLs para requisições do *SInteg*.

```
temperaturaURL: 'http://barbatheus.dashboard.com:4392/sinteg/temperature',
umidadeURL: 'http://barbatheus.dashboard.com:4392/sinteg/humidity',
portaURL: 'http://barbatheus.dashboard.com:4392/sinteg/reedswitch',
luzURL: 'http://barbatheus.dashboard.com:4392/sinteg/ldr',
```

Fonte: elaborado pelo autor

A Figura 41 apresenta o módulo do *SInteg* com as informações provindas das requisições.

Figura 41 – *Card* com as informações do *SInteg*.



Fonte: elaborado pelo autor

4.4.5 Problemas Enfrentados

Havia sido definido que dois módulos a mais seriam desenvolvidos, um do *Facebook* e outro do *Instagram*.

Quando foi estudado sobre cada uma das APIs e definido as informações que seriam obtidas, ambos mostrariam a quantidade de seguidores e as últimas menções do perfil ou página pré definida.

Entretanto, após algumas semanas quando seria desenvolvido estes módulos, houveram mudanças nestas APIs que impossibilitaram a continuação do trabalho.

Na API do *Facebook*, foi retirada a função que possibilitava a obtenção do número de seguidores, como mostrado na Figura 42.

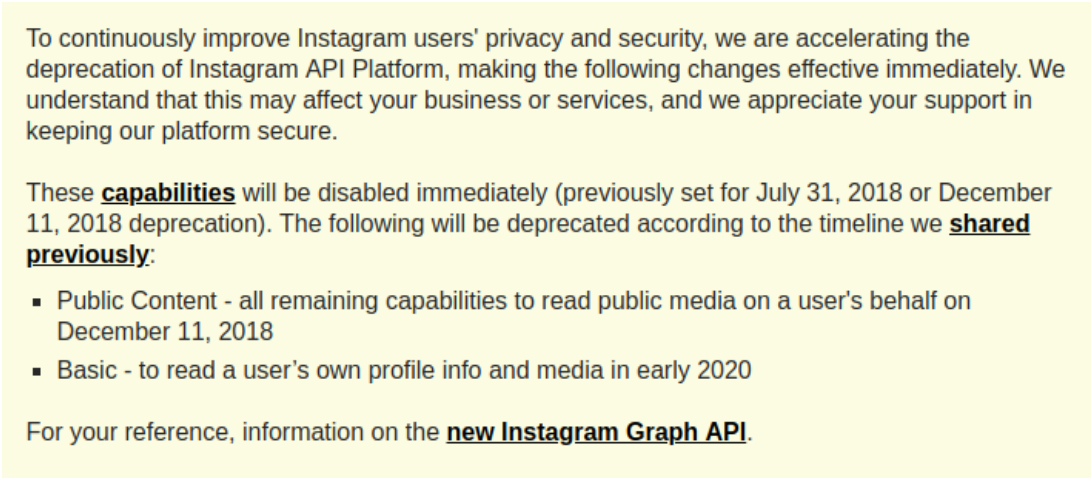
Figura 42 – Função retirada da documentação da API do *Facebook*.

Fonte: Facebook (2018)

Inicialmente, seria utilizado uma conta empresarial para obter os dados a serem mostrados no módulo, porém, para a mesma, seria requisitado a autenticidade da empresa, o que poderia gerar problemas futuros.

A API do *Instagram* permitia obter dados de contas não comerciais, entretanto as funções da mesma serão inutilizáveis no final deste ano, de acordo com a Figura 43.

Figura 43 – Informação sobre inutilização da API do *Instagram*.



Fonte: Instagram (2018)

5 CONCLUSÃO

Conforme definido, este projeto teve como objetivo o desenvolvimento de um *dashboard* modular para controle de projetos e ambientes.

O entendimento da importância da informação e sua interpretação foi garantido ao estudar diversos modelos de *dashboard* e a sua aplicação em diversos cenários, sendo estes marketing, executivo e até mesmo em *call centers*. O aprofundamente em painéis específicos para a área de computação foi essencial para a definição dos módulos a serem desenvolvidos.

Os *dashboard* existentes para a área de computação são utilizados principalmente para controle de servidores, com foco na parte de *hardware*, garantindo o controle de temperatura e umidade em salas de *datacenter*. O *dashboard* Spatie, que não possui foco em *hardware*, permite o controle de tarefas, a visualização de eventos futuros e também informa dados do áudio que está em reprodução no momento.

O *dashboard* desenvolvido reúne os principais *softwares* utilizados na área de computação. Além de permitir o controle de tarefas através de dados provindos do *Github*, também garante melhor comunicação e disseminação de informações através do *Slack*. Garante melhor controle do ambiente através da sua integração com o *SInteg* e com o *Spotify*.

Estas características não são visualizadas em outros projetos já existentes, já que a maioria dos mesmos são voltados para o controle de *hardware* e o painel mais similar apenas informa dados sobre o projeto que está sendo desenvolvido.

Com a melhor propagação da informação, que está disposta de uma maneira agradável e de fácil entendimento ao usuário, pode-se obter melhor gestão das informações e mais produtividade no trabalho realizado.

5.1 Trabalhos Futuros

Para trabalhos futuros propõe-se melhorias e novas funcionalidades ao projeto.

A utilização dos *cookies* não é a melhor maneira de realizar a troca de informações entre o servidor e o *dashboard* e para garantir mais segurança é proposto o desenvolvimento de um banco de dados para armazenamento de informações provindas das requisições realizadas. Informações como o *access token*, *client secret* e *client id* devem ser armazenadas de uma maneira segura para que informações sobre os dados do usuário não sejam expostas a terceiros.

Também é proposto o desenvolvimento dos dois módulos que não foram possíveis durante este projeto. Caso disponha-se de uma conta empresarial, o desenvolvimento dos mesmos é similar a todos os módulos em funcionamento no projeto.

A integração do *dashboard* com o *Facebook* e *Instagram* permitirá a visualização de informações sobre o marketing da empresa ou projeto. Com a análise dos dados pode-se verificar se a proposta utilizada é eficiente ou se mudanças são necessárias.

REFERÊNCIAS

AGUILHAR, L. *O Slack reúne alguns dos grupos mais legais da internet*. 2017. Disponível em: <https://link.estadao.com.br/blogs/ligia-aguilhar/slack-grupos-internet/>. Acesso em: 25 outubro 2018.

BULMA.IO. *Bulma Documentation*. 2017. Disponível em: <https://bulma.io/documentation/>. Acesso em: 20 setembro 2018.

CANALTECH. *O que é API?* 2017. Disponível em: <https://canaltech.com.br/software/o-que-e-api/>. Acesso em: 24 outubro 2018.

CARVALHO, J. *8 coisas que você precisa saber sobre dashboard*. 2016. Disponível em: <https://pt.linkedin.com/pulse/8_coisas_que_voce_precisa_saber_sobre_dashboard_jovenildo_carvalho>. Acesso em: 24 outubro 2018.

CHACON, B. S. S. *Pro Git*. 2. ed. Apress, 2014. Disponível em: <https://git-scm.com/book/en/v2>. Acesso em: 20 setembro 2018.

CISCO. *O que são cookies*. 2018. Disponível em: <https://www.cisco.com/c/pt-br/support/docs/security/web-security-appliance/117925-technote-csc-00.html>. Acesso em: 20 setembro 2018.

COADIC, Y.-F. L. *A ciência da informação*. 1. ed. [S.l.]: Brinquet de Lemos, 1996.

DEVFURIA. *O que é JavaScript?* 2018. Disponível em: <http://www.devfuria.com.br/javascript/o-que-e-javascript/>. Acesso em: 24 outubro 2018.

DIGITAL OCEAN. *Uma introdução ao OAuth 2*. 2018. Disponível em: <https://www.digitalocean.com/community/tutorials/uma-introducao-ao-oauth-2-pt>. Acesso em: 24 outubro 2018.

DRUBSCKY, L. *Marketing no Instagram: o guia para iniciantes*. 2018. Disponível em: <https://marketingdeconteudo.com/marketing-no-instagram/>. Acesso em: 25 outubro 2018.

E.MIX. *Dashboard: O que é e como funciona?* 2016. Disponível em: <https://emix.com.br/dashboard-o-que-e-e-como-funciona/>. Acesso em: 24 outubro 2018.

EXPRESSJS. *Express*. 2017. Disponível em: <https://expressjs.com/>. Acesso em: 12 setembro 2018.

FACEBOOK. *Graph API Reference*. 2018. Disponível em: <https://developers.facebook.com/docs/graph-api/reference/v3.1/user/subscribers>. Acesso em: 15 outubro 2018.

HERTEN, F. V. der. *Dashboard Spatie*. 2017. Disponível em: <https://github.com/spatie/dashboard.spatie.be>. Acesso em: 09 setembro 2018.

HOSTINGER. *O que é CSS?* 2018. Disponível em: <https://www.hostinger.com.br/tutoriais/o-que-e-css-guia-basico-de-css>. Acesso em: 24 outubro 2018.

IMASTERS. *Como funciona o protocolo OAuth 2.0*. 2017. Disponível em: <https://imasters.com.br/desenvolvimento/como-funciona-o-protocolo-oauth-2-0>. Acesso em: 09 setembro 2018.

IN8. *Consultoria em TI: O que é Switch e para que serve*. 2018. Disponível em: <https://in8.com.br/consultoria-em-ti-o-que-e-switch-e-para-que-serve/>. Acesso em: 24 outubro 2018.

INSTAGRAM. *Instagram Developers*. 2018. Disponível em: <https://www.instagram.com/developer>. Acesso em: 15 outubro 2018.

JAVASCRIPT INFO. *An Introduction to Javascript*. 2018. Disponível em: <https://javascript.info/intro>. Acesso em: 24 outubro 2018.

KINAST, P. *Saiba quais são as 15 Redes sociais mais usadas no mundo*. 2018. Disponível em: <https://optclean.com.br/15-redes-sociais-mais-usadas-no-mundo/>. Acesso em: 25 outubro 2018.

MIOZZO, J. *Rede social github facilita comunição entre programadores e está crescendo*. 2015. Disponível em: <https://startse.com/noticia/rede_social_github_facilita_comunicao_entre_programadores_e_est_crescendo>. Acesso em: 25 outubro 2018.

MOZILLA. *Express Web Framework*. 2017a. Disponível em: <https://developer.mozilla.org/pt-BR/docs/Learn/Server_side/Express_Nodejs>. Acesso em: 12 setembro 2018.

MOZILLA. *Express/Node Introdution*. 2017b. Disponível em: <https://developer.mozilla.org/en-US/docs/Learn/Server-side/Express-Nodejs/Introduction>. Acesso em: 20 setembro 2018.

MOZILLA. *O que é um servidor web (web server)?* 2018. Disponível em: <https://developer.mozilla.org/pt-BR/docs/Learn/Common_questions/o_que_e_um_web_server>. Acesso em: 05 maio 2018.

NODEJS. *About NodeJS*. 2017. Disponível em: <https://nodejs.org/en/about/>. Acesso em: 12 setembro 2018.

NPM. *dotenv*. 2018. Disponível em: <https://www.npmjs.com/package/dotenv>. Acesso em: 25 outubro 2018.

OAUTH. *OAuth 2.0 Grant Types*. 2018. Disponível em: <https://oauth.net/2/grant-types/>. Acesso em: 24 outubro 2018.

OP SERVICES. *O que é um dashboard? O guia completo e definitivo!* 2017. Disponível em: <https://www.opservices.com.br/o-que-e-um-dashboard/>. Acesso em: 19 março 2018.

OPSERVICES. *Indicadores de Performance e Modelos de Dashboard por área da empresa*. 2016. Disponível em: <https://www.opservices.com.br/dashboards-de-indicadores-de-performance/>. Acesso em: 24 outubro 2018.

PRATES, G. *Bulma: framework CSS baseado em flexbox*. 2016. Disponível em: <https://tableless.com.br/bulma-framework-css-baseado-em-flexbox/>. Acesso em: 24 outubro 2018.

SASS. *Sass Documentation*. 2018. Disponível em: <https://sass-lang.com/documentation>. Acesso em: 12 setembro 2018.

SCHMITZ, D. *Tudo que você queria saber sobre Git e GitHub, mas tinha vergonha de perguntar*. 2015. Disponível em: <https://tableless.com.br/tudo_que_voce_queria_saber_sobre_git_e_github_mas_tinha_vergonha_de_perguntar/>. Acesso em: 24 outubro 2018.

SERRA, J. P. *Manual de Teoria da Comunicação*. 2. ed. [S.l.]: Livros Labcom, 2007.

SIGNIFICADOS. 2011. Disponível em: <https://www.significados.com.br/website/>. Acesso em: 05 maio 2018.

SIMEAO, G. R. *Uma abordagem inicial sobre a biblioteca VueJs*. 2017. Disponível em: <https://medium.com/trainingcenter/vuejs_init_uma_abordagem_inicial_sobre_a_biblioteca_javascript_vuejs_b2490771f05c>. Acesso em: 24 outubro 2018.

SLACK. *OAuth and Permissions*. 2018a. Disponível em: <https://api.slack.com/apps>. Acesso em: 25 outubro 2018.

SLACK. *API Methods*. 2018b. Disponível em: <https://api.slack.com/methods>. Acesso em: 25 outubro 2018.

SPOTIFY. *Web Playback SDK*. 2018. Disponível em: <https://developer.spotify.com/documentation/web-playback-sdk/>. Acesso em: 24 outubro 2018.

SPOTIFY. *o que é o Spotify?* 2018b. Disponível em: <https://support.spotify.com/br/using-spotify/the-basics/what-is-spotify/>. Acesso em: 25 outubro 2018.

TELECORP. *Data center*. 2017. Disponível em: <https://www.telecorp.com.br/glossario/data-center/>. Acesso em: 24 outubro 2018.

VERTIGO TECNOLOGIA. *O que é API? Entenda de uma maneira simples*. 2018. Disponível em: <https://vertigo.com.br/o-que-e-api-entenda-de-uma-maneira-simples>. Acesso em: 05 maio 2018.

VUEJS. *Vue Documentation*. 2017. Disponível em: <https://vuejs.org/v2/guide/>. Acesso em: 12 setembro 2018.

VUEJSBR. *Por que Vuejs é uma boa opção*. 2016. Disponível em: <http://vuejs-brasil.com.br/por-que-vuejs-e-uma-boa-opcao/>. Acesso em: 12 setembro 2018.

W3SCHOOLS. *HTML5 Tutorial*. 2018. Disponível em: <https://www.w3schools.com/html/>. Acesso em: 12 setembro 2018.

W/MORAIS. *As diferenças entre site estático e site dinâmico*. 2016. Disponível em: <https://www.wmorais.com.br/single-post/2016/07/12/As_diferenças_entre_site_estático_e_site_dinâmico>. Acesso em: 24 outubro 2018.