INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DA BAHIA

CAMPUS VITÓRIA DA CONQUISTA

BACHARELADO EM SISTEMAS DE INFORMAÇÃO

FAGNER PINHEIRO SANTOS

Desenvolvimento de um SaaS Multiplataforma utilizando o Modelo Multi-Tenancy no Contexto de Computação em Nuvem

Vitória da Conquista-BA
2018

FAGNER PINHEIRO SANTOS

Desenvolvimento de um SaaS Multiplataforma utilizando o Modelo Multi-Tenancy no Contexto de Computação em Nuvem

Trabalho monográfico apresentado ao Programa de Graduação em Bacharelado em Sistemas de Informação do Instituto Federal de Educação, Ciência e Tecnologia da Bahia *Campus* Vitória da Conquista como requisito parcial para a obtenção do grau de Bacharel em Sistemas de Informação.

Orientador: Me. Pablo Freire Matos

Vitória da Conquista-BA
2018

S237d Santos, Fagner Pinheiro.
Desenvolvimento de um SaaS multiplataforma utilizando o modelo Multi-Tenancy no contexto de computação em nuvem / Fagner Pinheiro Santos. - Vitória da Conquista, BA, 2018.
64 f.

Orientador: Prof. Me. Pablo Freire Matos.

Trabalho de Conclusão de Curso (Graduação) Bacharelado em Sistemas de Informação - Instituto Federal de Educação, Ciência e Tecnologia da Bahia – *Campus* de Vitória da Conquista-BA, 2018.

1. Computação em Nuvem. 2. Bancos de Dados. 3. API RESTful. 4. Multitenancy. 5. Software – Serviço I. Santos, Fagner Pinheiro. II. Título.

CDD : 005.758

Elaborada pela Biblioteca do Instituto Federal de Educação, Ciência e Tecnologia da Bahia - *Campus* Vitória da Conquista-BA. Bibliotecária - Rita Fonseca CRB - 5 / 1747.

FAGNER PINHEIRO SANTOS

Desenvolvimento de um SaaS Multiplataforma utilizando o Modelo Multi-Tenancy no Contexto de Computação em Nuvem

Este Trabalho de Conclusão de Curso foi julgado adequado para a obtenção do grau de Bacharel em Sistemas de Informação, e aprovado em sua forma final pelo Programa de Graduação em Bacharelado em Sistemas de Informação do Instituto Federal de Educação, Ciência e Tecnologia da Bahia *Campus* Vitória da Conquista.

Vitória da Conquista, 26 de abril de 2018.

Prof. Me. Mailson Sousa Couto
(Coordenador do Curso - IFBA *Campus* Vitória da Conquista)

Banca Examinadora:

Prof. Me. Pablo Freire Matos
(Orientador - IFBA *Campus* Vitória da Conquista)

Prof. Me. Crescêncio Rodrigues Lima Neto
(IFBA *Campus* Vitória da Conquista)

Prof. Me. Luis Paulo da Silva Carvalho
(IFBA *Campus* Vitória da Conquista)

Prof. Me. Stênio Longo Araújo
(IFBA *Campus* Vitória da Conquista e UESB)

À minha mãe, que lutou contra tudo e todos para que eu chegasse até aqui e que, mesmo longe fisicamente e cheia de saudade, me manteve focado nos meus objetivos e nunca me deixou sozinho, sempre me aconselhando e me dando forças.

AGRADECIMENTOS

À Deus, primeiramente, toda a gratidão, sabedoria e força que me proporcionou durante toda esta caminhada.

À minha mãe, Valdete, que nunca mediu esforços e moveu montanhas para que eu pudesse realizar minha graduação e que, apesar de longe, sempre se fez presente das mais diversas maneiras, meu eterno amor.

Ao meu pai, Antônio, que sempre se orgulhou das minhas conquistas e nunca me deixou desamparado. Aos meus avós minha eterna gratidão por todos os conselhos.

À minha namorada, melhor amiga e futura esposa, Ana Paula, por todo o amor, ajuda e apoio a mim dedicado, sempre me ouvindo, aconselhando e me dando força para que eu seguisse em frente e que, junto com Arnaldera, sempre estiveram presentes para me alegrar e me confortar.

Aos meus irmãos, Pedro e Samilla, em especial ao meu irmão Anderson, que sempre me apoiaram em quaisquer decisões. Aos meus sobrinhos Lara, João, Vinicius, Maria, Laís e Miguel por terem sido fonte de alegrias e brincadeiras. Aos meus cunhados Leila, Bany, Paty e Duda e meus sogros Seu Paulo e Dona Albene por me acolherem e me escutarem sempre quando precisei.

Aos meus tios (Valmar, Eliana, Socorro e Valdemar) e primos por todo o incentivo e apoio.

À Matheus Peixoto e Joice Telles por me acolherem em sua casa quando vim morar em Vitória da Conquista, me ajudando a crescer e criar responsabilidade e, além de tudo, sempre estarem presentes e me ajudando a resolver todos os problemas que tive.

Aos meus amigos de infância, em especial, a Cleyver, Henrique, Lucas, Lúcio, Nilson e Edson por nunca se afastarem, apesar de toda a distância e falta de tempo. À Seu Edson e Dona Sirlene por sempre manterem um carinho tão grande por mim.

Aos meus colegas de curso por todos os projetos realizados (ou não) e que me proporcionaram tanto conhecimento, em especial, a Randler, Ramon, Rafael, Patrick, Italo e Allexandre.

À minha primeira professora de programação, Roberta Gondim, por me aconselhar a vir morar em Vitória da Conquista e sempre acreditar no meu potencial, meu muito obrigado.

Aos meus professores de graduação, por tanto conhecimento compartilhado ao longo de todos estes anos, em especial, aos professores, Crescêncio Lima, Mailson Couto, Luis Paulo e Fernando Cardeal.

Ao meu orientador, Pablo Matos, por me guiar ao longo deste trabalho e sempre me cobrar para que eu me esforçasse mais, mantendo sempre meu foco no projeto.

Finalmente, agradeço a todos que, direta ou indiretamente, me ajudaram durante minha graduação e na realização deste projeto.

"Um herói pode ser qualquer um, até mesmo um homem fazendo algo tão simples e reconfortante como colocar um casaco em torno dos ombros de um menino, para deixá-lo saber que o mundo não tinha terminado."

(Batman, o cavaleiro das trevas ressurge)

RESUMO

Com o avanço e o crescimento da Internet, muitas empresas remodelaram a forma de entrega de seus *softwares* fazendo com que estes sejam adquiridos, configurados e acessados diretamente pela Internet, reduzindo os custos da aplicação tanto para o cliente quanto para a empresa, além de facilitar a implantação, já que não será necessário adquirir servidores para executarem as aplicações. Este modelo de entrega é denominado *Software como Serviço* (*SaaS*) e vem em constante crescimento nos últimos anos. Porém, para conseguir reduzir ainda mais os custos operacionais, muitas empresas optam por utilizar um mesmo banco de dados para diversos clientes, o que pode acarretar em permitir que dados sejam visualizados por usuários que não os pertence, um erro grave na segurança destes dados. Para contornar este problema, essas empresas optam por utilizar o modelo *multi-tenancy*, responsável por garantir que os dados só sejam acessados por seus proprietários, mesmo estando no mesmo banco de dados. Sendo assim, este trabalho tem como objetivo principal o desenvolvimento e implantação de um *SaaS* para gestão de clínicas fisioterapêuticas. Para isso, foram analisadas as tecnologias existentes no mercado para o desenvolvimento desse tipo de aplicação e como elas se comunicam entre si. Após a implantação, foi realizado uma pesquisa nas empresas que utilizaram a aplicação, fazendo com que a hipótese levantada neste estudo fosse comprovada, sendo possível determinar que um *software* como serviço utilizando um único banco de dados em conjunto com o modelo *multi-tenancy* atende às demandas das aplicações, e garante segurança e integridade dos dados tão bem quanto *softwares* com bancos isolados para cada cliente.

Palavras-chave: Computação em Nuvem. Bancos de Dados. *Back-end. RESTful*. *Multi-tenancy*. *Software* como Serviço.

ABSTRACT

With the advancement and growth of the Internet, many companies have reshaped the way their *software* is delivered by having them purchased, configured and accessed directly over the Internet, reducing application costs for both the customer and the company, as well as facilitating because you will not have to buy servers to run the applications. This delivery model is called Software as a Service (SaaS) and has been constantly growing in recent years. However, in order to further reduce operational costs, many companies choose to use the same database for multiple clients, which may result in data being viewed by users who do not belong to them, a serious error in the security of this data. To circumvent this problem, these companies choose to use the *multi-tenancy* model, which is responsible for ensuring that the data is only accessed by its owners, even though it is in the same database. Therefore, this work has as main goal the development and implementation of a SaaS for the management of physiotherapeutic clinics. For this, we analyzed the existing technologies in the market for the development of this type of application and how they communicate with each other. After the implementation, a survey was carried out in the companies that used the application, making the hypothesis raised in this study to be proven, being possible to determine that a *software* as a service using a single database in conjunction with the *multi-tenancy* model meets the demands of most applications and ensures data security and integrity as well as *software* with isolated databases for each customer.

Keywords: Cloud Computing. Databases. Back-end. RESTful. Multi-tenancy. Software as a Service.

LISTA DE FIGURAS

LISTA DE QUADROS

SUMÁRIO

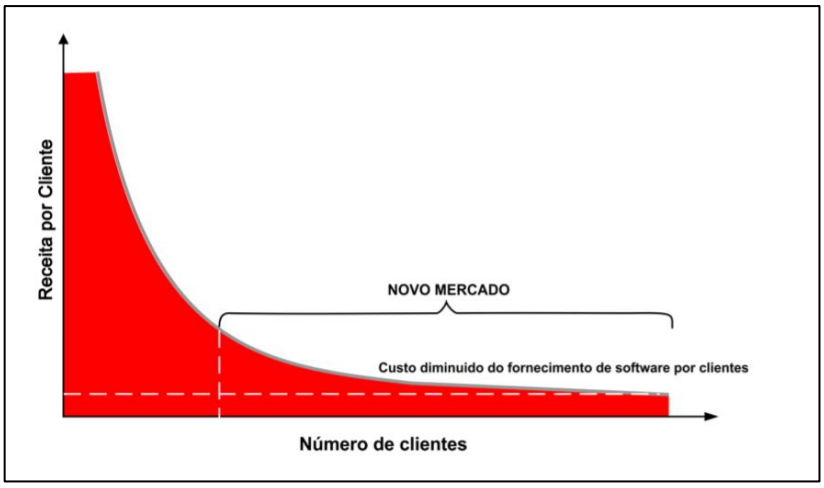
1 INTRODUÇÃO

Segundo Barnes (2014), com a crescente evolução das telecomunicações e o advento das conexões banda larga, o modelo de serviço *Software* como Serviço (SaaS) começou a tornar-se viável. Barnes (2014) cita ainda que prova deste crescimento é que, segundo a *International Data Corporation* (IDC), o mercado de aplicações baseado em computação em nuvem mundial crescerá de US$ 47,7 bilhões em 2013 para US$ 107 bilhões em 2017.

De acordo com Mesquita (2016), o SaaS é um modelo de negócio, no qual a comercialização, a distribuição do *software* e as soluções digitais são realizadas diretamente por meio da Internet. Nada é instalado diretamente na máquina do cliente e toda a aplicação fica disponível através de um navegador de Internet. Isto faz com que esta máquina possa ser acessada de qualquer parte do mundo, sendo muito vantajoso quando comparado ao modelo tradicional de *software,* no qual a aplicação depende de um servidor físico na empresa e é instalada diretamente no computador do cliente.

No mercado de *software* existem duas abordagens para a venda e entrega de *software*: uma é vender uma aplicação para um pequeno grupo de clientes que estão dispostos a pagar um alto valor por este produto; a outra é vender (ou prover um serviço) a um valor bem mais baixo, desde que atenda a um número grande de clientes com a mesma aplicação. Esse cenário, como pode ser visto na Figura 1, indica que quanto mais um provedor de serviços consegue baixar o custo de um produto aos seus clientes, maior será o número de clientes que poderá ser alcançado (NETO, 2012).

Figura 1: Aumento do número de clientes em aplicações mais baratas.



Fonte: Neto (2012).

Neste contexto, surgiu o conceito da Cauda Longa que, segundo Neto (2012), é um fenômeno observado em empresas de Internet que conseguem faturar com produtos de nicho de mercado mais que os tradicionais produtos da “moda”. De acordo com Neto (2012), este processo se tornou viável com o advento da Internet, já que a inexistência de limitação do espaço físico para a exibição de produtos faz com que os produtos que atendem uma pequena fatia do mercado sejam exibidos da mesma forma que os produtos focados em um público geral.

Segundo Chong e Carraro (2006), fornecedores de soluções complexas de *software* se defrontam com uma curva de mercado semelhante, no qual apenas empresas com alto poder aquisitivo tendem a adquirir seus produtos. Esses *softwares* normalmente requerem servidores dedicados, atendimento personalizado, instalação no local e manutenções periódicas no local físico da empresa, o que faz com que o custo para fornecer esse tipo de aplicação seja um tanto quanto alto.

Segundo Mesquita (2016), o SaaS trás consigo diversas vantagens. Dentre elas se destacam o pequeno valor de investimento para a implantação de um novo *software* na empresa e a facilidade nas atualizações, já que, conforme dito anteriormente, por funcionar totalmente por meio da Internet, não é necessário nenhuma instalação ou atualização em servidores físicos do cliente.

Sendo assim, uma quantidade enorme de pequenas e médias empresas que poderiam se beneficiar dessas soluções não possuem os recursos para adquiri-las. Assim, no modelo SaaS de fornecimento de *software*, têm-se focado em desenvolver soluções e infraestruturas de baixo custo e com alto aproveitamento de recursos durante o uso dessas aplicações por um grande número de usuários. Isto faz com que os custos destes recursos sejam cada vez menores, o que diminui, também, os preços cobrados pelas aplicações.

A partir do exposto, apesar das grandes vantagens das aplicações SaaS, como garantir a integridade e o isolamento dos dados sabendo que todos os recursos são compartilhados e, se este isolamento não for feito corretamente, a chance de que os dados de um usuário sejam vistos por outros usuários é considerável? Para isso, este trabalho objetiva propor a utilização do conceito de *multi-tenancy* para o isolamento dos dados em servidores e bancos de dados compartilhados em aplicações SaaS no contexto da computação em nuvem.

1.1 Motivação

Sabe-se que a busca por diminuição dos custos operacionais é constante, independente da área. Pensando no desenvolvimento *de software*, é perceptível, que uma aplicação SaaS tende a ser muito mais barata e viável para a maior parte das empresas, visto que não é necessário a compra de servidores por parte do cliente e todos os recursos da aplicação são compartilhados entre os mesmos (NETO, 2012). Porém, devido a esse compartilhamento de recursos, o isolamento dos dados é um problema constante, visto que, se não for feito corretamente, os dados poderão ser vistos por qualquer usuário do sistema. Muitas empresas optam por utilizar servidores de banco de dados isolados para cada cliente, o que pode acarretar em perda de recursos e aumento nos custos operacionais.

Além disso, com o crescimento da Internet e a grande quantidade de sistemas operacionais disponíveis no mercado, para garantir o sucesso de uma aplicação, esta deve ser desenvolvida pensando em atender a todas as plataformas possíveis. Para isso, uma das soluções é o desenvolvimento de uma aplicação *web*, já que seu acesso depende apenas de um dispositivo que possua conexão com Internet e um *browser* instalado.

Portanto, para garantir a integridade e o isolamento dos dados utilizando uma única instância de um banco de dados e a disponibilidade desta aplicação para o maior número de plataformas possível, é necessário que seja utilizada tecnologias que garantam que um dado só será consumido pelo seu proprietário, independente se estes utilizam o mesmo banco alinhado às tecnologias de desenvolvimento *web* para disponibilizar a aplicação através da Internet.

1.2 Justificativa

Empresas brasileiras estão cada vez mais adeptas às soluções de TI no modelo SaaS. Segundo uma pesquisa feita pela Capgemini no Brasil, o SaaS é o modelo mais utilizado para entregar serviços pela nuvem, com 92% das empresas adotando pelo menos uma solução com aplicações deste tipo (SAASHOLIC, 2017).

É evidente o crescimento do SaaS no mercado de *softwares*. Porém, um dos grandes problemas encontrados no desenvolvimento de aplicações deste tipo é a forma como o isolamento e a segurança dos dados acontecerão. Além disso, grande parte das aplicações disponíveis não possui suporte a diversas plataformas (*web* e *mobile*), o que dificulta seu acesso e disponibilidade para seus clientes.

Com o intuito de indicar a utilização mais adequada e propor uma solução para contornar este problema, recomenda-se a utilização do modelo *multi-tenancy* para garantir que os dados só serão vistos por seus respectivos proprietários, mesmo que estes dados estejam compartilhados no mesmo banco. Ademais, possibilitar a utilização de tecnologias *web* para o desenvolvimento das aplicações, o que faz com que estas sejam acessadas por qualquer dispositivo que possua acesso à Internet e um *browser*.

1.3 Problema

Com base nos dados apresentados, este trabalho procura responder ao seguinte questionamento: o uso do modelo *multi-tenancy* aplicado no desenvolvimento do *Software* como Serviço (SaaS) no contexto de computação em nuvem pode contribuir com a redução de custo, a melhoria no desempenho, a segurança e a integridade dos dados?

1.4 Objetivos

A seguir são listados os objetivos (geral e específicos) a serem atingidos com o desenvolvimento deste trabalho.

1.4.1 Objetivo Geral

Desenvolver uma aplicação SaaS web multiplataforma utilizando os conceitos de *multi-tenancy* no contexto de computação em nuvem.

1.4.2 Objetivos Específicos

Como objetivos específicos, estão os seguintes:

- Propor a utilização de modelo *multi-tenancy* em uma arquitetura de desenvolvimento de *software* como serviço aplicado no contexto de computação em nuvem;
- Instanciar uma aplicação SaaS multiplataforma utilizando os conceitos de *multi-tenancy*, a fim de validar os fatores que influenciam a adoção do uso de *multi-tenancy* em aplicações SaaS.

1.5 Hipótese

Uma aplicação SaaS utilizando os conceitos de *multi-tenancy* garante a segurança, a integridade e o isolamento dos dados mesmo utilizando um único

banco de dados para vários clientes e um único servidor de aplicação para os vários acessos de diversos clientes.

1.6 Metodologia

1.6.1 Classificação da Pesquisa

O trabalho a ser desenvolvido, segundo Gil (2010), tem caráter predominantemente qualitativo, uma vez que pretende desenvolver uma aplicação SaaS multiplataforma utilizando os conceitos de *multi-tenancy* aplicados à computação em nuvem. Após o desenvolvimento, o sistema será implantado em uma clínica fisioterapêutica a fim de validar seu propósito.

1.6.2 Procedimentos Metodológicos

A realização deste trabalho se dividiu em etapas. Inicialmente, foi realizado um levantamento dos trabalhos referentes ao tema para que pudesse verificar o estado atual das pesquisas na área. Os parâmetros para a realização do estudo foram delineados, e também foram definidas as tecnologias, os dados, as ferramentas e os *softwares*.

Após a definição do escopo, foram definidos os requisitos para o desenvolvimento do projeto, bem como quais tecnologias seriam utilizadas no *software*. Por fim, foi feito a implementação do sistema e o *deploy* deste para um servidor, tornando a aplicação disponível através da Internet.

1.7 Organização do Trabalho

Este Trabalho de Conclusão de Curso está organizado em 06 (seis) capítulos distribuídos na seguinte ordem: Capítulo 1: foram apresentados a motivação, a justificativa, o problema, os objetivos (geral e específicos), a hipótese e a metodologia deste trabalho; Capítulo 2: são apresentadas as áreas de conhecimento relevantes para este trabalho; Capítulo 3: são apresentados os trabalhos que possuem área de pesquisa similar ao projeto proposto neste trabalho; Capitulo 4: é apresentado o modelo proposto para o desenvolvimento do *software* e as tecnologias existentes para utilização; Capítulo 5: é apresentado a instanciação do modelo *multi-tenancy* em um sistema real no domínio médico (clínica fisioterapêutica); e Capítulo 6: são apresentadas as conclusões e proposto os trabalhos futuros para a continuação do projeto.

2 FUNDAMENTAÇÃO TEÓRICA

A seguir são discutidos os principais temas relacionados a este trabalho com o intuito de facilitar para o leitor um entendimento sobre o problema abordado. Sendo assim, na Seção 2.1 é contemplada a definição de Computação em Nuvem. Na Seção 2.2 são introduzidos os conceitos de Software como Serviço (SaaS). Na Seção 2.3 são discutidos os assuntos relacionados à *Multi-tenancy,* modelo utilizado para isolamento dos dados de diversas empresas em um mesmo banco de dados. Já na Seção 2.4 são discutidos os assuntos relacionados ao Back-end da aplicação. Por fim, na Seção 2.5 são descritas as tecnologias que serão utilizadas neste trabalho.

2.1 Computação em Nuvem

Segundo o *National Institute of Standards and Technology* (MELL; GRANCE, 2011), computação em nuvem é um modelo que permite acesso configurável a diversos recursos computacionais e que são fornecidos com o mínimo de esforço ou interação. Sendo assim, a computação em nuvem provê à utilização dos recursos computacionais de forma compartilhada e interligados por meio da Internet.

Ainda segundo o NIST, a computação em nuvem é composta por cinco características essenciais (MELL; GRANCE, 2011):

- Alocação de recursos sob demanda: O usuário não precisa adquirir todos os recursos computacionais oferecidos. Essa aquisição deve ser feita de forma unilateral, e utilizar apenas o necessário;
- *Pooling* de recursos: Os recursos do provedor são agrupados de forma que seja possível atender vários consumidores e devem ser alocados de acordo com a demanda do consumidor. Estes recursos incluem o armazenamento, o processamento, a memória, a largura de banda, o sistema operacional e as máquinas virtuais;
- Amplo acesso a rede: Recursos estão disponíveis através da rede e podem ser acessados por qualquer dispositivo com acesso à Internet;
- Elasticidade rápida: A aquisição de novos recursos deve ser feita de forma rápida e elástica e, em alguns casos, automaticamente,

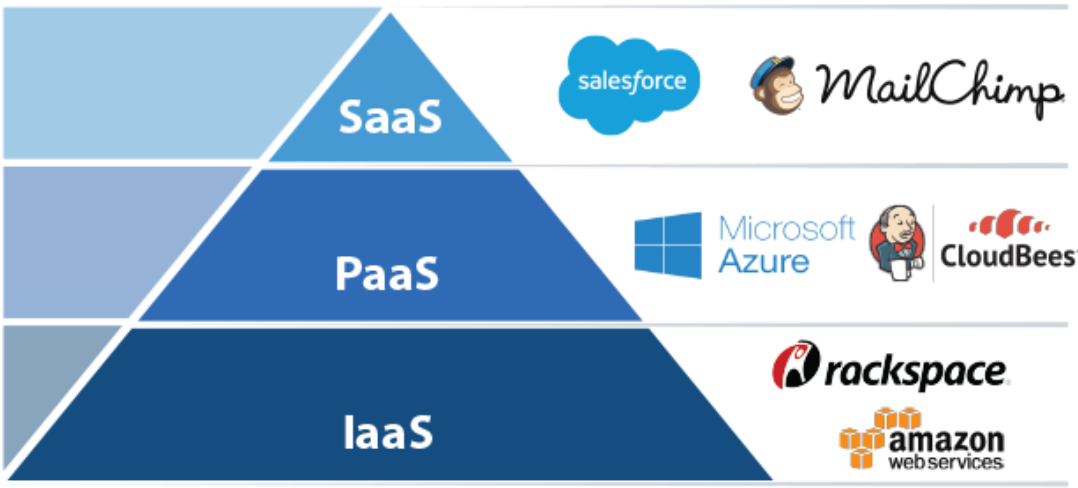
caso haja a necessidade de um aumento de uso em horários de pico;

- Serviço medido: Os sistemas devem controlar e otimizar, automaticamente, a utilização dos recursos. O uso de recursos pode ser monitorado, controlado e relatado ao consumir deste serviço.

O NIST afirma que a computação em nuvem é dividida em três modelos de serviço (Figura 2) (MELL; GRANCE, 2011):

- Software como Serviço (SaaS): Este modelo de serviço tem como foco o usuário final e a capacidade fornecida por ele é a de usar aplicações do fornecedor em uma infraestrutura da nuvem. Estas aplicações são acessíveis por vários dispositivos clientes que tenham acesso à Internet e um *browser* web. Este modelo não permite que o consumidor administre ou controle a infraestrutura básica do serviço;
- Plataforma como Serviço (PaaS): A capacidade fornecida ao consumidor deste tipo de modelo é a de realizar *deploy* de uma aplicação em uma infraestrutura predefinida. O consumidor não administra ou controla a infraestrutura neste modelo. Porém, tem controle sobre as aplicações hospedadas nestes servidores;
- Infraestrutura como Serviço (IaaS): Neste modelo de serviço o consumidor tem total controle sobre os recursos computacionais para a implantação das aplicações que serão hospedados nos servidores, porém não administra ou controla a infraestrutura de nuvem subjacente.

Figura 2: Classificação das aplicações SaaS.



Fonte: Muspratt (2017).

Neste trabalho será dado como foco principal os conceitos relacionados à SaaS, pois é neste contexto em que o *multi-tenancy* é empregado.

2.2 SaaS

Segundo Asaas (2013), o SaaS (*Software* como Serviço) é um programa que somente é utilizado através da Internet, não sendo necessário a instalação para sua utilização por parte do cliente. O crescimento do SaaS veio junto com a popularização da Internet. Antes dos anos 90, toda empresa que necessitava de um *software* precisava comprar sua licença e criar servidores próprios para o armazenamento destes dados. Porém, segundo Asaas (2013), as empresas estavam ficando insatisfeitas com os custos elevados para manter um *software*, exigindo do mercado de TI um modelo mais viável para a compra e utilização de programas.

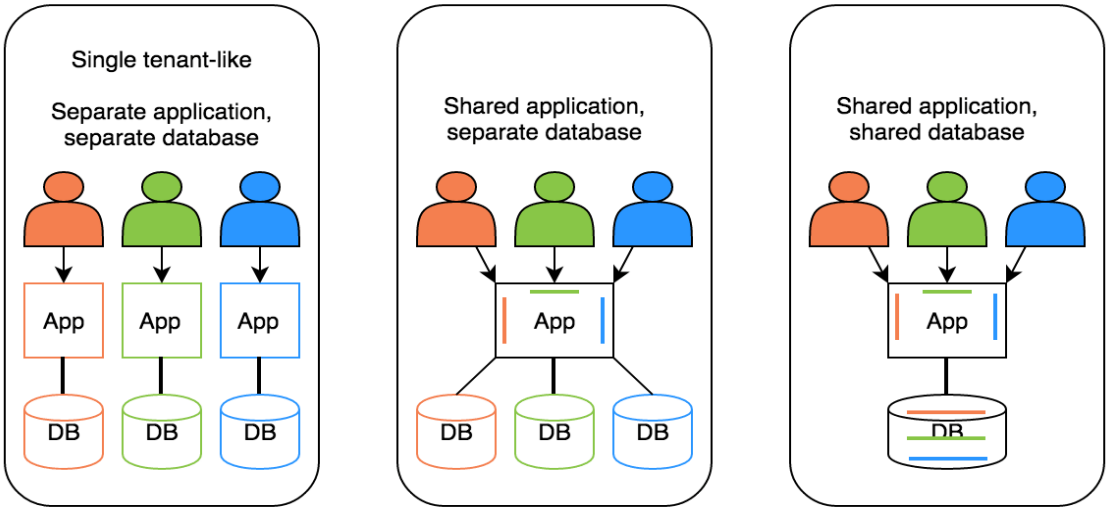
Segundo Harris e Ahmed (2011), existem três tipos de aplicações que utilizam o modelo SaaS:

- *Multi-instance*: Neste tipo, as aplicações são executadas em ambientes nos quais o servidor é compartilhado. Sendo assim, uma cópia da aplicação é configurada para cada cliente e implantada em um mesmo servidor web;
- *Single-instance*: As aplicações são executadas para um único cliente em um tipo de serviço exclusivo. Esta abordagem é considerada um grande desperdício, já que um servidor pode executar com apenas 10% da sua capacidade;

- *Multi-tenancy*: Provê uma única aplicação compartilhada por vários clientes. Assim, várias aplicações “virtuais” são criadas na mesma instância do servidor.

Pensando em melhorar a utilização de recursos e diminuir os eventuais gastos relacionados à manutenção e implantação dos sistemas, observou-se que o modelo *multi-tenancy* é o mais adequado para ser utilizado neste trabalho. Na Figura 3 é ilustrado a diferença entre os tipos de aplicações SaaS.

Figura 3: Classificação das aplicações SaaS



Fonte: Otero (2017).

2.3 Multi-tenancy

No modelo tradicional de aplicações de *software*, cada organização precisa possuir um servidor de banco de dados exclusivo em sua empresa. Os recursos utilizados para a utilização de uma aplicação podem ficar caros e, muitas vezes, ociosos. Sendo assim, surge o conceito de *multi-tenancy* que, segundo Zaidman (2010), são aplicações que permitem a otimização no uso dos recursos de *hardware* através do compartilhamento de instâncias da aplicação e dos servidores de banco de dados, e permitem a configuração para atender às necessidades de cada cliente como se estivessem sendo executadas em ambientes isolados.

Segundo Neto (2012), o *tenant* é a entidade organizacional que aluga uma aplicação *multi-tenancy,* fornecida através do modelo *SaaS.* Cada *tenant* pode ter vários usuários, chamados de *stakeholders* da organização.

Nesse sentido, uma aplicação *multi-tenancy* deve garantir que os dados de cada cliente se mantenham de forma isolada, não sendo visíveis para outros clientes, mesmo estando em um único ambiente isolado. É essencial que cada

tenant tenha a impressão de que está utilizando uma aplicação com recursos dedicados.

2.4 Back-end

Segundo Pires (2017), é comum a utilização de aplicações que funcionem exclusivamente pela Internet independente de plataforma, sendo consumidas por navegadores em *desktops*, *notebooks* ou dispositivos móveis. Por outro lado, grandes empresas estão sempre alimentando seus *softwares* de gestão com novos dados. Porém, como permitir que aplicações em diversas plataformas consigam consumir os dados de um mesmo banco de dados e, assim, compartilhar informações?

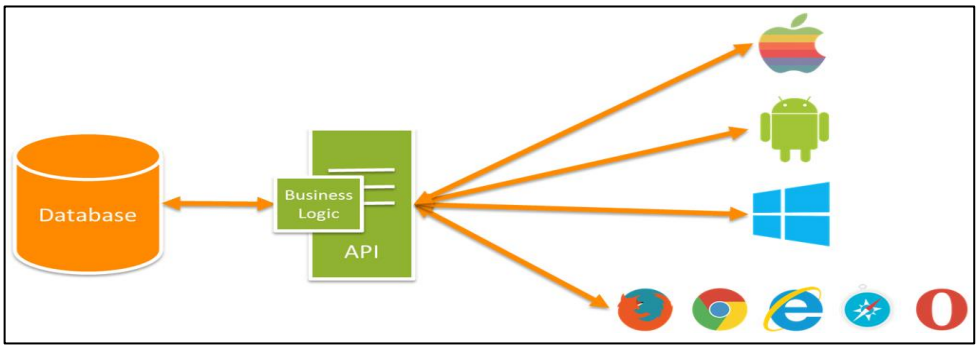
Pires (2017) afirma que, durante anos, diversas alternativas surgiram e, de uma forma geral, esse tipo de aplicação ficou conhecida como APIs (Interface de Programação de Aplicações). Basicamente, o funcionamento dessas aplicações baseia-se em fornecer um ponto central de acesso entre a aplicação e o cliente, seja ele um usuário ou uma aplicação.

Segundo Ribeiro (2016), uma API é como uma chave de tradução para que 2 aplicativos conversem entre si. Por exemplo, é como um intérprete que ouve uma pessoa (aplicação 1) falando grego e traduz para o português para que outra pessoa (aplicação 2) consiga reconhecer as informações enviadas.

Em outras palavras, uma API é responsável por fornecer um ponto central de comunicação para aplicações de diversas plataformas. Seu principal objetivo é prover um caminho único para que as aplicações consigam acessar o mesmo banco de dados de forma centralizada e ajuda a armazenar toda a regra de negócio da aplicação (Figura 4).

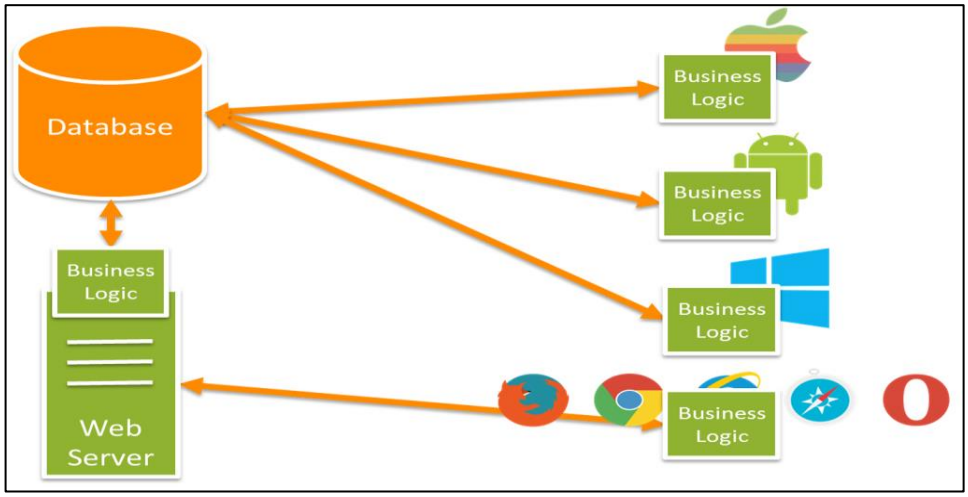
Pode-se perceber, então, a grande utilidade de uma API, na qual normalmente cada aplicação possui sua lógica de negócio e a comunicação entre o banco de dados é feita de forma independente (Figura 5).

Figura 4: Aplicações utilizando APIs.



Fonte: Kearn (2015).

Figura 5: Aplicações sem utilizar API.



Fonte: Kearn (2015).

2.5 Tecnologias de Desenvolvimento

Para o desenvolvimento do *software,* foram utilizadas as tecnologias *HTML* para a criação da estrutura das páginas do sistema; *CSS*, utilizada na estilização das páginas; PHP e JavaScript, utilizadas para implementar a lógica de negócio da aplicação; Laravel, *framework* para facilitar o desenvolvimento *web* utilizando o PHP; MySQL, utilizado como sistema de gerenciamento de banco de dados para armazenamento das informações do sistema; Vue.js, *framework* para criação de aplicações *front-end* utilizando *JavaScript.*

A Linguagem para Marcação de Hipertextos (HTML) foi desenvolvida com o intuito de interligar hipertextos através de *links* globais, denominados *hiperlinks*. Criada por Tim Barners-Lee, seu objetivo era facilitar o compartilhamento de pesquisas entre os cientistas espalhados pelo mundo (SILVA, 2014).

A HTML é uma linguagem baseada em marcação, cujos elementos são demarcados para que seja definido quais seus significados. Serve, principalmente, para a criação da estrutura e esqueleto das páginas disponíveis na Internet.

Dentro da estrutura do HTML, é possível adicionar parâmetros que façam com que as *tags* sejam estilizadas e, assim, criar páginas personalizadas. Porém, para cada elemento que queira personalizar, um novo código deve ser escrito, o que faz com que a quantidade de código repetido fique cada vez maior. Assim, a *World Wide Web Consortium* (W3C) criou a *Cascade Style Sheets* (CSS).

O CSS é uma especificação que define como os elementos que compõe uma página serão exibidos (MATERA, 2012). Segundo Matera (2012), a forma correta de publicar uma página *web* é seguir uma estrutura semântica, na qual o CSS ficará responsável por definir todas as propriedades do *layout* da página (cores, tamanho de fonte, dentre outros) e o HTML ficará responsável por fornecer a "arquitetura" da página.

Tendo o HTML e o CSS para criar a estrutura das páginas *web* disponíveis na Internet e com a expansão da Internet, logo houve a necessidade de implementar lógica às páginas *web* e, assim, criar sistemas complexos que seriam executados através da Internet. Para isso, diversas linguagens de programação foram desenvolvidas, dentre elas o PHP.

A linguagem de programação PHP, desenvolvida por Rasmus Lerdorf, tem o objetivo de dinamizar páginas na *web*, o que faz com que elas se modifiquem em tempo de execução. Criada em 1995, atualmente está na versão 7.1.0. (DALL'OGLIO, 2015).

Por ter sido desenvolvida com foco exclusivo ao suporte de aplicações *web*, o PHP é uma das principais linguagens de desenvolvimento em aplicações desta categoria. Além disso, a linguagem possui suporte a diversos sistemas gerenciadores de bancos de dados de forma nativa e suporte a protocolos de comunicação, como o HTTP e o POP3.

Com as inúmeras vantagens das aplicações *web,* o crescimento dos *softwares* utilizando esta arquitetura cresceu de forma exponencial, o que fez com que as aplicações ficassem cada vez maiores, e isto dificultava o desenvolvimento e acabava atrasando o início do desenvolvimento de um novo sistema. Assim, inúmeros *frameworks* foram desenvolvidos, pensando em facilitar e organizar o desenvolvimento de aplicações. *Laravel* é um exemplo de *framework* para a linguagem PHP.

O *Laravel*, atualmente em sua versão 5.5, é um *framework* focado em desenvolvimento simples e eficiente utilizando o PHP. Criado por Taylor Otwell, o

Laravel é um dos principais *frameworks* da atualidade e provê diversos recursos que facilitam o desenvolvimento de aplicações *web*, como um sistema de *templates*, rotas e comunicação nativa com diversos bancos de dados (DIAS, 2014).

Para concentrar grandes volumes de dados em um local centralizado para, posteriormente, serem acessados por diversos usuários, são utilizados os bancos de dados (BD), definidos por Elmasri e Navathe (2005) como “uma coleção lógica e coerente de dados com algum significado inerente. Uma organização de dados ao acaso não pode ser corretamente interpretada como um banco de dados”.

Porém, esse grande volume de dados deve ser gerenciado de forma simples e eficiente, evitando que os dados armazenados sejam de difícil acesso. Para isso, existem os Sistemas de Gerenciamento de Banco de Dados (SGBD), conjunto de *software* capaz de armazenar e gerenciar dados armazenados fisicamente.

Há diversos SGBDs existentes atualmente, dentre eles o MySQL, um dos principais SGBDs do mundo. O MySQL utiliza a linguagem de consulta SQL, possui código aberto e foi desenvolvido em 1995 pela empresa sueca MySQL AB e posteriormente comprada pela *Sun Microsystem*. Atualmente se encontra na versão 5.4, e é utilizado por diversas empresas de tecnologia em todo o mundo (PISA, 2012).

O *Vue.js* é um *framework JavaScript* para criação de interfaces iterativas na *web*. Seu foco é no desenvolvimento de interfaces melhoradas e adaptáveis e na criação de componentes reutilizáveis, que podem ser definidos como blocos de código HTML, CSS e JavaScript que podem ser reaproveitados em diversas partes do sistema (REIS, 2016).

3 TRABALHOS CORRELATOS

Neste capítulo são apresentados trabalhos que possuem relação com o projeto proposto.

3.1 Sistema para Gestão de Confinamento de Gado de Corte

De acordo com Bellei (2016), a indústria brasileira de carne bovina mostra-se expressiva mundialmente, visto que em 2015 a cadeia bovinocultura de corte no Brasil atingiu um faturamento de R$ 5,9 bilhões. Ainda segundo o autor, o principal responsável por estes números é a pecuária de corte. Os animais ficam dispostos em lotes dentro de uma área restrita para receber alimentação e assistência, com o intuito de aumentar suas massas corporais para o abate, promovendo o máximo de aproveitamento de seu potencial.

Sendo assim, pensou-se em um sistema computacional que permitisse o registro e cruzamento dos dados obtidos diariamente a fim de abstrair as informações que facilitariam a organização, o planejamento e a administração do estabelecimento como um todo. Bellei (2016) afirma que este sistema deveria funcionar de forma responsiva, multiplataforma e utilizar as tecnologias de Computação em Nuvem.

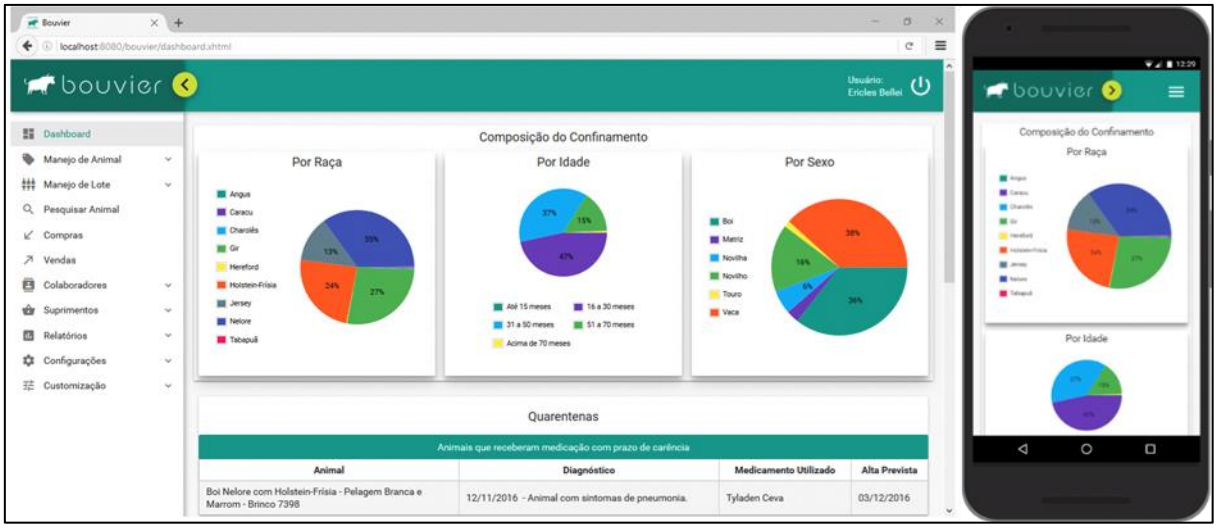
Ainda segundo o autor, a principal motivação para o projeto é a carência de uma ferramenta auxiliar específica que dificulta o acompanhamento das atividades, tornando menos eficiente a tarefa de gerenciamento do estabelecimento e torna mais difícil o monitoramento do rebanho.

Por fim, o autor relata que o *software* denominado *Bouvier (*Figura 6*)* cumpre os objetivos propostos e que a solução desenvolvida como um aplicativo *web* hospedado na nuvem emerge como tendência e desafia muitas concepções tradicionais da área porque faz repensar a forma de organização e programação, e que promete aumento de produtividade aliado à economia de tempo e recursos.

O *software* desenvolvido pelo autor utiliza tecnologia Java para o desenvolvimento da aplicação. Porém, o uso do *multi-tenancy* não é notado, diferente do proposto neste trabalho, o que pode dificultar a utilização do *software* por mais de um cliente, já que um servidor de banco de dados dedicado seria necessário para o novo usuário da aplicação. Outro ponto a se destacar é a não utilização de uma API RESTful, o que faz com que o *software* não consiga prover

informações para outros clientes e que, apesar de possuir responsividade, pode não funcionar tão bem em dispositivos móveis.

Figura 6: Página principal do *software* Bouvier.



Fonte: Bellei (2016).

3.2 Aplicação Multi-tenancy com Hibernate Shards

A ideia proposta por Custódio e Júnior (2012) é de uma aplicação utilizando os conceitos de *multi-tenancy* aplicados à tecnologia *Hibernate Shards*. Segundo os autores, a principal motivação para esta abordagem se deve ao fato de que, com o avanço da computação em nuvem, empresas de diferentes tamanhos têm passado por mudanças no fornecimento de *software* aos seus clientes, sendo a maior adoção o fornecimento do *Software* como Serviço (*SaaS*). Porém, segundo Custódio e Junior (2012), implementar o *multi-tenancy* sem a ajuda de um *framework* pode se tornar uma tarefa difícil, o que os fez escolher pela utilização do *Hibernate Shards*.

Segundo Custódio e Junior (2012), a principal motivação do uso do *multi-tenancy* é do compartilhamento de recursos no servidor por vários clientes, evitando que seja necessário o uso de um servidor dedicado por sistema, o que reduz muito os custos operacionais do *software*. Os autores afirmam ainda que um *software multi-tenancy* deve garantir que nenhum dado deve “vazar”, apenas o proprietário da informação poderá visualizá-la.

Por fim, os autores relatam que foi possível perceber a importância da pesquisa de novas tecnologias e conceitos, e que o fornecimento de um *software* como serviço pode trazer muitas vantagens para empresas de desenvolvimento. Ainda segundo os autores, foi possível concluir que o *Hibernate Shards* atende às

necessidades para o desenvolvimento de solução *multi-tenancy* para pequenas aplicações.

3.3 Estudo de Mapeamento Sistemático de Aplicações Multi-tenancy

Segundo Neto (2012), os *Softwares* como serviço (SaaS) estão se tornando uma forma dominante de entrega de *software* aos usuários, já que dentre diversas vantagens, a diminuição dos preços de manutenção e vendas são as mais percebidas. Para chegar a estas conclusões, o autor realizou um mapeamento sistemático que identificou o estado da arte do *multi-tenancy*, propostas de implementação e principais lacunas existentes nesta área.

De forma geral, o mapeamento sistemático abrange os principais conceitos utilizados em SaaS e demonstram a real necessidade do uso do *multi-tenancy* neste tipo de aplicação.

O autor afirma ainda que o crescimento desse tipo de aplicação só tende a aumentar, haja vista que as tecnologias estão em constante crescimento e a tendência é que exista cada vez mais *softwares* como serviço e que estes sejam mais acessíveis.

Por fim, o autor conclui que nos últimos anos muitas empresas têm saído do modelo de entrega de *software* tradicional para o modelo de fornecimento via web e, cada vez mais, essas aplicações se tornam um serviço que pode ser adquirido e utilizado exclusivamente através da Internet.

Apesar de não ser focado no desenvolvimento de uma aplicação, mas no mapeamento sistemático de aplicações *multi-tenancy*, o estudo contribuiu muito para o trabalho desenvolvido por apresentar as vantagens do *multi-tenancy*.

3.4 Sistemas Multi-tenancy

Segundo Col (2015), em qualquer área de negócio a inovação é sempre um diferencial, pois proporciona às empresas mais dinamismo e vantagens competitivas. Segundo a autora, com o intuito de agregar inovação e otimizar processos aos estabelecimentos de entrega em domicílio, o trabalho propôs o desenvolvimento de um sistema *web* de controle de entregas que será utilizado por usuários que possuem diversos dispositivos. Portanto, o *software* deveria se adaptar a diferentes tamanhos de tela e, para isso, foi utilizado o *framework front-end Bootstrap* em conjunto com o HTML. Já na parte de desenvolvimento do *back-end,* cujo foco é na lógica de negócio da aplicação, foi utilizado o *framework Spring*.

Segundo a autora, a principal motivação para o desenvolvimento deste projeto (Figura 7) é que, pela falta de tempo, muitas pessoas encontraram na entrega em domicílio a melhor forma de se alimentar sem ter que enfrentar filas, falta de estacionamento, e outros fatores. A facilidade do acesso à *web*, em especial à crescente *web mobile*, tornou possível a realização dessa tarefa de forma rápida e precisa.

Figura 7: Listagem de produtos cadastrados.

Fonte: Col (2015).

Sendo então uma aplicação que vários clientes iriam utilizá-la de maneira compartilhada, o uso do modelo *multi-tenancy* foi imprescindível, visto que seria impossível dedicar um único servidor para cada cliente da aplicação.

Por fim, Col (2015) relata que é possível afirmar que as tecnologias utilizadas possuem um grande impacto nos quesitos de qualidade de produtividade, visto que a implementação do sistema ocorreu em poucos meses e o resultado atingiu todas as expectativas.

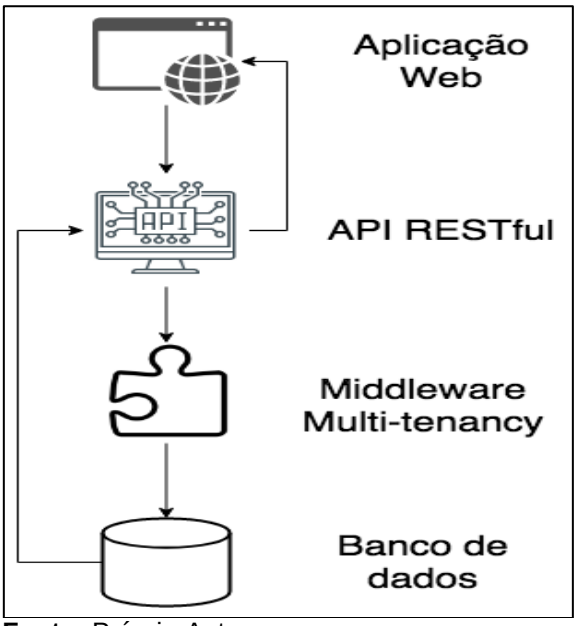
Por se tratar de um *software* para pedidos em diversos restaurantes, o projeto realizado pela autora utiliza o modelo *multi-tenancy*. Porém, não provê um serviço RESTful para disponibilizar informações para outros clientes, o que pode prejudicar em sua utilização, já que é bem comum que pedidos em restaurantes sejam feitos em dispositivos móveis.

4 PROPOSTA DE UTILIZAÇÃO DE MODELO MULTI-TENANCY

Este capítulo objetiva apresentar o modelo a ser utilizado no desenvolvimento de aplicações utilizando o *multi-tenancy* em conjunto com uma API RESTful para prover informações a diversos clientes. O funcionamento do modelo utilizado é composto por quatro componentes (Figura 8): Aplicação Web (Componente 1); API RESTful (Componente 2); *Middleware Multi-tenancy* (Componente 3); e Banco de Dados (Componente 4).

O Componente 1 é responsável pela iteração com o usuário, sendo que a entrada de dados é fornecida pelo usuário e as informações processadas são devolvidas. O Componente 2 é responsável por prover um ponto central de acesso aos dados do banco de dados. Este componente é de grande importância, pois é ele quem obtém a requisição enviada pelo Componente 1, obtém os dados através do banco de dados e devolve ao usuário. O Componente 3 é responsável por capturar todas as requisições feitas através do Componente 2 e devolver apenas dados pertencentes ao usuário que os requisitou. Finalmente, o Componente 4 é responsável por armazenar todas as informações de todos os clientes e usuários que utilizam o sistema.

Figura 8: Modelo *Full Stack*.



Fonte: Próprio Autor.

No componente Aplicação Web, os dados são exibidos para os usuários do sistema. Seu principal papel é a interação com o usuário, pois é a partir deste

componente que é identificado se o usuário quer exibir algum registro ou inserir um novo registro no banco de dados.

A partir da iteração feita pelo usuário na Aplicação *Web*, é feita uma requisição para o componente seguinte, a *API RESTful*. Este componente é responsável por definir a lógica de negócio da aplicação, a saber: como os dados devem ser tratados durante o fluxo da aplicação.

Em seguida, antes dos dados serem requisitados ou inseridos no banco de dados da empresa, deve ser verificado qual usuário está realizando tal ação para que os dados sejam inseridos ou exibidos de acordo com seu proprietário. Para isso, o *Middleware Multi-tenancy* verificará qual usuário está logado na aplicação e retornará apenas dados deste usuário.

Quando um usuário solicitar um determinado dado por meio da Aplicação Web, a requisição feita através da API RESTful passará pelo *Middleware Multi-tenancy* que verificará se aquele dado pertence ao usuário que o solicitou e, caso positivo, permitirá que o dado seja exibido. Sempre que um usuário realizar requisições de inserção de novas informações no banco, o *Middleware Multi-tenancy* relacionará esta informação ao usuário que realizou a requisição, o que faz com que todo dado possua um proprietário. Por fim, o banco de dados é responsável por armazenar todos os dados de todas as empresas que utilizem o *software*.

A seguir será explicado detalhadamente o funcionamento dos quatros componentes.

4.1 Aplicação Web

O componente Aplicação Web é responsável por capturar todas as interações dos usuários com o sistema e enviá-las ao componente API RESTful. Sendo assim, este componente deverá ser capaz de se comunicar e trocar informações com a API utilizando a notação JSON. Para isso, há diversas tecnologias e *frameworks* no mercado que possuem suporte nativo a essa troca de informações. A principal linguagem utilizada é o JavaScript, como pode ser visto no Quadro 1.

Quadro 1: Principais tecnologias para aplicações web.

Tecnologia	Linguagem	Descrição
Angular	TypeScript	*Framework* JavaScript mais utilizado no mundo. Possui uma gama de componentes e tecnologias para criação de aplicações web *front-end* e possui suporte nativo a requisições HTTP e troca de informações utilizando o formato JSON.
Vue.js	JavaScript	*Framework open-source* com principal foco a criação de *web-components*. Possui também suporte nativo a requisições HTTP e troca de mensagens no formato JSON.
React.js	JavaScript	Desenvolvido pelo time de desenvolvedores do Facebook, o React é um *framework* também focado no desenvolvimento de aplicações *front-end* e possui uma grande quantidade de adeptos pelo mundo todo.

Fonte: Próprio Autor.

Devido à quantidade de tecnologias existentes, deve-se verificar os pontos positivos e negativos de cada uma antes de decidir-se pela escolha em um projeto a ser desenvolvido. No Quadro 2 é possível visualizar as principais características de cada uma das tecnologias citadas no Quadro 1.

Quadro 2: Características das tecnologias *front-end*.

Característica	Angular	React	Vue.js
Estável	Sim	Sim	Sim
Comunidade ativa	Sim	Sim	Sim
Boa documentação	Sim	Sim	Sim
Fácil aprendizado	Não com TypeScript	Meio difícil	Sim
Integração com Bootstrap	Sim	Sim	Sim
Tamanho	566K	139K	58.8K
Reuso de código	Sim	Não, somente CSS	Sim
Velocidade de código	Lento	Normal	Rápido
Reatividade	Meio difícil	Sim	Sim
Baseado em componentes	Sim	Sim	Sim

Fonte: Belitsoft (2018)**.**

O *framework* Vue foi o escolhido para ser utilizado no desenvolvimento do projeto por ser um *framework* robusto, estável, com ótima documentação e com curva de aprendizagem muito grande.

4.2 API RESTful

A API RESTful é responsável por comportar toda a lógica de negócio da aplicação, o que faz com que esta lógica só precise ser feira uma única vez e todas as aplicações clientes só entendam quais dados devem enviar para a API. Seu principal papel é servir como um caminho único e central entre as aplicações cliente e o banco de dados. Para o desenvolvimento da API, há diversas linguagens de programação e *frameworks* disponíveis no mercado (Quadro 3).

Quadro 3: Principais tecnologias para criação de APIs RESTful.

Tecnologia	Linguagem	Descrição
ASP.NET Web API	C#	*Framework* C# com foco em desenvolvimento de APIs RESTful sobre a plataforma .NET. Criado pela Microsoft, possui comunidade ativa e utilizado por grandes empresas.
Laravel	PHP	*Framework* PHP para desenvolvimento *web* mais utilizado no mundo. Sua principal premissa é o desenvolvimento rápido de aplicações com um código limpo e uma grande reutilização de código.
Jersey	Java	*Framework* Java para desenvolvimento de serviços RESTful, possui código fonte *open-source* e é muito utilizado na comunidade Java de desenvolvimento.
Django	Python	*Framework* Python mais utilizado no mundo. *Open-source* e com uma comunidade muito ativa, possui diversas funcionalidades para o desenvolvimento ágil, com uma grande reutilização de código e facilidade de desenvolvimento.

Fonte: Próprio Autor.

Para o desenvolvimento do projeto foi utilizado o *framework* Laravel pela quantidade de usuários ativos na comunidade e conhecimento mais amplo.

4.3 Middleware Multi-tenancy

Antes de inserir ou requisitar uma informação ao banco de dados, é necessário verificar se aquele dado pertence ao usuário que o está requisitando ou até relacionar a informação a ser inserida com este usuário. Para isso, é necessário a utilização de um *middleware* que interceptará a requisição feita pela API e adicionará essas informações de controle. Por ser uma camada extra desenvolvida no *back-end*, o *middleware* deverá ser desenvolvido utilizando a mesma tecnologia usada no desenvolvimento do serviço *RESTful.*

4.4 Banco de Dados

O banco de dados é responsável por armazenar todas as informações da aplicação e deixá-las disponíveis sempre que necessário. É a última parte da aplicação e uma das mais importantes, pois seu foco principal é o armazenamento

dos dados e estes não podem ser perdidos ou armazenados de forma incorreta. Há diversos sistemas gerenciadores de bancos de dados disponíveis no mercado (Quadro 4).

Quadro 4: Principais tecnologias para bancos de dados.

Tecnologia	Linguagem	Descrição
MySQL	SQL	Sistema de gerenciamento de banco de dados que utiliza a linguagem SQL para inserção de manipulação de dados. Adquirido pela Oracle e atualmente um dos SGBDs mais populares do mundo com mais de 10 milhões de instalações.
PostgreSQL	SQL	SGBD de código aberto e mantido pela comunidade, considerado um dos SGBDs mais avançados do mundo. Desenvolvido nas linguagens C, Perl e Sh, possui suporte a multiplataforma e possui diversos adeptos espalhados pelo mundo.
MariaDB	SQL	Após a aquisição do MySQL pela Oracle, o MariaDB surgiu como uma alternativa ao MySQL, sendo desenvolvido pelo mesmo criador. Possui código fonte aberto e é o principal substituto do MySQL.

Fonte: Próprio Autor.

No projeto foi utilizado o banco de dados MySQL para armazenar os dados dos usuários que utilizam o sistema.

5 INSTANCIAÇÃO DO MODELO *MULTI-TENANCY*

Neste capítulo será exemplificado o uso do modelo *multi-tenancy* no desenvolvimento de uma aplicação de gestão de clínicas fisioterapêuticas, o que demonstra a utilização da proposta descrita no Capítulo 4 em um projeto real.

5.1 Escopo do Sistema

O sistema tem como objetivo gerenciar pequenas e médias clínicas de fisioterapia, desde o cadastro de funcionários e pacientes até o histórico das consultas que o paciente realizou dentro da clínica. O desenvolvimento da aplicação é dividido em duas etapas. A primeira é a criação de uma API RESTful utilizando o *framework Laravel* responsável por definir toda a lógica de negócio da aplicação. Esta aplicação tem como foco disponibilizar um serviço para que as aplicações clientes que necessitem acessar dados do banco de dados não façam este acesso diretamente ao BD, mas sim à API que fará este trabalho.

A segunda é a criação de uma aplicação *front-end* utilizando o *framework Vue.js* responsável por consumir os dados da API e, assim, obter as informações existentes no banco de dados da empresa. Esta aplicação tem como foco o usuário e objetiva disponibilizar um meio de acesso simples e de fácil interação para que o usuário seja capaz de manipular os dados do banco de dados.

5.2 Modelagem

Nesta seção são apresentados os passos para a modelagem do sistema proposto, desde o levantamento dos requisitos até a criação do diagrama de entidade-relacionamento.

5.2.1 Requisitos Funcionais

No Quadro 5 são listados os Requisitos Funcionais (RF) identificados para o sistema.

Quadro 5: Requisitos funcionais.

Identificação	Nome	Descrição
RF01	Cadastrar pacientes	Permitir ao usuário secretário o cadastro de pacientes.
RF02	Cadastrar funcionários	Permitir ao usuário administrador o cadastro de funcionários junto com seu nível de acesso.
RF03	Cadastrar avaliação postural do paciente	Permitir ao usuário fisioterapeuta o cadastro de avaliações posturais de um determinado paciente.
RF04	Cadastrar avaliação fisioterapêutica do paciente	Permitir ao usuário fisioterapeuta o cadastro de avaliações fisioterapêuticas de um determinado paciente.
RF05	Exibir avaliações do paciente	Permitir ao usuário fisioterapeuta a visualização de todas as avaliações de determinado paciente.

Fonte: Próprio Autor.

5.2.2 Requisitos Não Funcionais

No Quadro 6 é contemplado os Requisitos Não Funcionais (RNF) do sistema relacionado ao uso da aplicação como um todo.

Quadro 6: Requisitos não funcionais.

Identificação	Nome	Descrição
RNF01	Usabilidade	O sistema deve ser de simples e fácil acesso por parte dos usuários.
RNF02	Portabilidade	O sistema deverá ser executado em qualquer sistema operacional que possua um navegador de Internet instalado, independente do tamanho da tela do dispositivo.
RNF03	Segurança	Os dados pessoais dos pacientes devem ser criptografados a fim de garantir uma maior segurança e só poderão ser acessados por usuários previamente permitidos e autenticados.
RNF04	Confiabilidade	Os dados de um usuário só poderão ser visualizados por ele mesmo, não por terceiros.

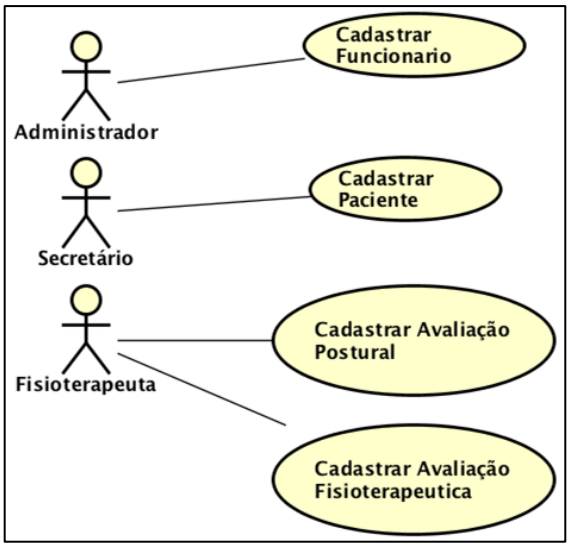
Fonte: Próprio Autor.

5.2.3 Diagrama de Casos de Uso

No diagrama de casos de uso apresentado na Figura 9 são ilustradas as funcionalidades essenciais do sistema, utilizadas pelos atores administrador, secretário e fisioterapeuta. O usuário Administrador é responsável por cadastrar novos funcionários no sistema. O secretário é responsável por cadastrar novos pacientes no sistema. O funcionário fisioterapeuta é responsável por cadastrar as avaliações posturais e fisioterapêuticas dos pacientes previamente cadastrados no sistema e alimentar seu histórico.

Um resumo dos fluxos básicos e alternativos da aplicação encontra-se no Quadro 7. A descrição completa encontra-se no Apêndice A – Descrição dos Casos de Uso.

Figura 9: Diagrama de Casos de Uso.



Fonte: Próprio Autor.

Quadro 7: Resumo dos fluxos básicos da aplicação.

Caso de Uso	Descrição
UC01 – Cadastrar Empresa	Para cadastrar uma empresa no sistema, deve-se informar os seguintes dados: nome, e-mail e CNPJ. Em seguida, deve-se clicar no botão “Cadastrar”. Após isso, o sistema criará um usuário com a senha “123456789” que será utilizada para autenticar o usuário “mestre” na aplicação.
UC02 – Autenticar Usuário	Para autenticar um usuário no sistema, deve-se informar seus dados de e-mail e senha na tela de login da aplicação e apertar o botão “Entrar”. Após isso, o sistema validará esses dados e realizará a autenticação do usuário, redirecionando-o para a tela inicial.
UC03 – Cadastrar Funcionário	Quando um funcionário é contratado na empresa, o administrador deve cadastrá-lo no sistema. Para isso, o administrador deve acessar o menu “funcionários” e selecionar a opção “Cadastrar”. Após isso, deve-se preencher o formulário de cadastro e clicar na opção “Salvar”. O sistema verificará se os dados são validos e realizará o cadastro do novo funcionário.
UC04 – Cadastrar Paciente	Para cadastrar um paciente no sistema, o usuário secretário deverá ir até o menu “pacientes” e selecionar a opção “Cadastrar”. Após isso, deve-se preencher o formulário de cadastro e clicar na opção “Salvar”. O sistema verificará se os dados são válidos e realizará o cadastro do novo paciente.
UC05 – Cadastrar Avaliação Postural	Para cadastrar a avaliação postural de um paciente, o usuário fisioterapeuta deve selecionar o menu “Paciente”, selecionar a opção “Pesquisar Paciente”, digitar o nome completo do paciente e entrar no perfil do mesmo. Após isso, selecionar a opção “Cadastrar nova avaliação postural” e preencher o formulário. Ao fim do preenchimento, clicar na opção “Salvar” para que os dados sejam enviados ao banco de dados. O sistema verificará se os dados são válidos e realizará o cadastro da avaliação postural do paciente desejado.

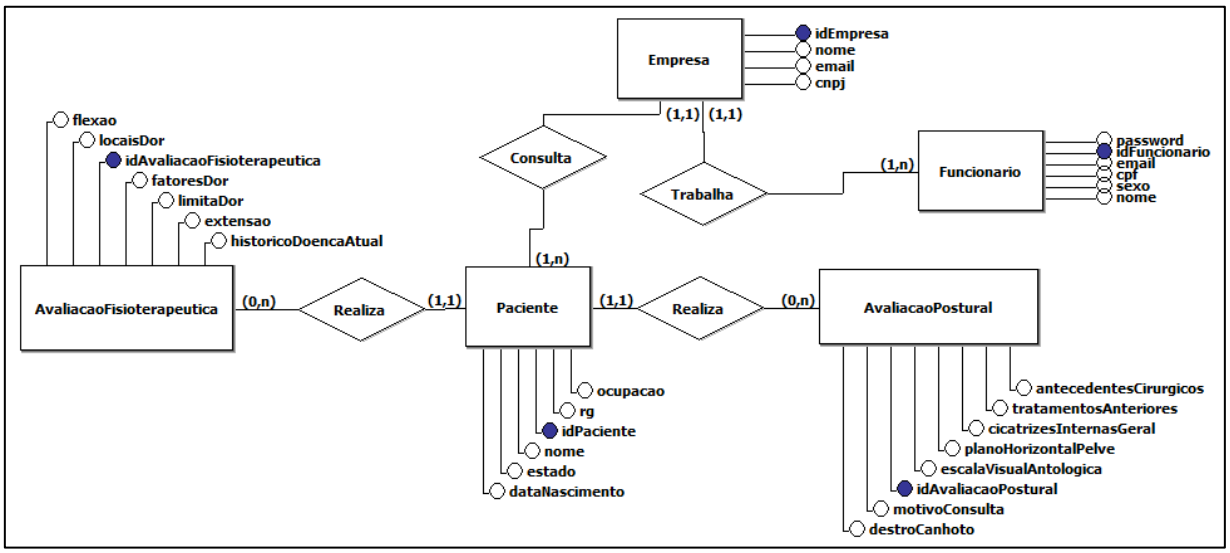
UC06 – Cadastrar Avaliação Fisioterapêutica	Para cadastrar a avaliação fisioterapêutica de um paciente, o usuário fisioterapeuta deve selecionar o menu “Paciente”, selecionar a opção “Pesquisar Paciente”, digitar o nome completo do paciente e entrar no perfil do mesmo. Após isso, selecionar a opção “Cadastrar nova avaliação fisioterapêutica” e preencher o formulário. Ao fim do preenchimento, clicar na opção “Salvar” para que os dados sejam enviados ao banco de dados. O sistema verificará se os dados são válidos e realizará o cadastro da avaliação fisioterapêutica do paciente desejado.

Fonte: Próprio Autor.

5.2.4 Diagrama Entidade-Relacionamento

Na Figura 10 é ilustrado o diagrama de entidade-relacionamento, em sua forma resumida, responsável por representar a estrutura do banco de dados do sistema proposto. A versão completa do diagrama pode ser encontrada no repositório na Internet[1].

Figura 10: Diagrama de Entidade-Relacionamento.



Fonte: Próprio Autor.

A entidade 'Empresa' é responsável por armazenar os dados das empresas que irão utilizar o sistema. Esta entidade é considerada “central”, pois é através desta que será possível identificar quais dados pertencem a qual empresa, o que evita que diferentes informações sejam exibidas para diferentes empresas.

A entidade 'Funcionario' é responsável por armazenar as informações dos funcionários que trabalham em determinada empresa, além das credenciais necessárias para que seja possível realizar o *login* no sistema (*e-mail e password)*. Todo funcionário deve se relacionar com a empresa que o mesmo trabalha. Sendo assim, a entidade 'Funcionario' possui um relacionamento com a entidade 'Empresa'

[1] Disponível em: https://goo.gl/MrzE1K

através da relação 'Trabalha', o que faz com que um funcionário trabalhe em apenas uma empresa. Porém, uma empresa pode possuir diversos funcionários relacionados a ela.

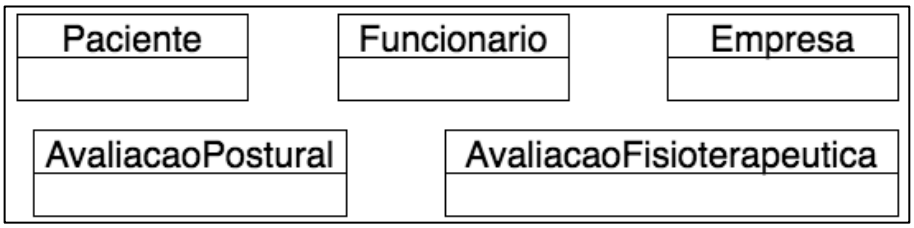
As informações pessoais dos pacientes serão armazenadas na entidade 'Paciente', na qual os dados pessoais deverão ser salvos. Esta entidade também possui relação com a entidade 'Empresa', já que um paciente somente poderá se relacionar com uma clínica por vez.

Sabe-se que os pacientes poderão realizar dois tipos de avaliações: a avaliação postural e avaliação fisioterapêutica. Cada avaliação deve ser mantida no sistema e, caso necessário, uma nova avaliação deve ser relacionada com o paciente. Assim, as entidades 'AvaliacaoPostural' e 'AvaliacaoFisioterapeutica' são responsáveis por armazenar as informações dessas avaliações.

5.2.5 Arquitetura

O *software* foi desenvolvido com a utilização do padrão arquitetural *MVC (Model, View, Controller)*, mesmo padrão empregado no *framework Laravel.* A camada Model é responsável por definir os atributos e relacionamentos que cada classe possuirá. As classes existentes no sistema são Paciente, Funcionario, Empresa, AvaliacaoPostural e AvaliacaoFisioterapeutica. Na Figura 11 é exibida uma versão resumida do Model para uma melhor visualização. O diagrama de classe completo da camada Model está disponível no repositório online[2].

Figura 11: Camada *Model* resumida (sem atributos e métodos).

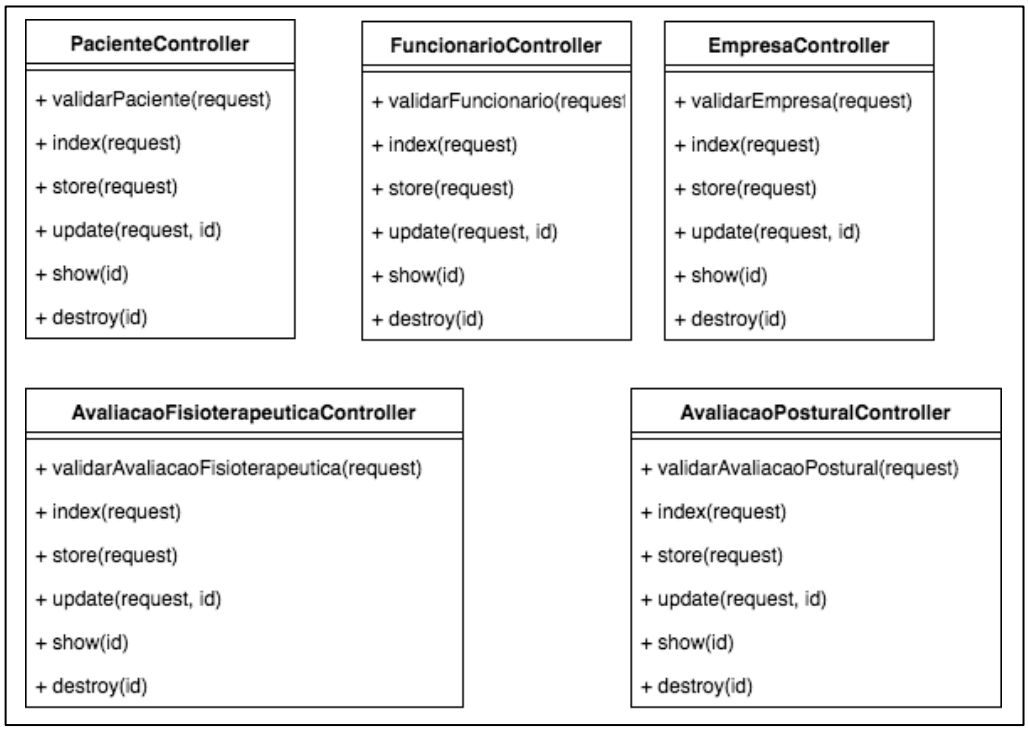


Fonte: Próprio Autor.

O *Controller* é responsável por definir a lógica de negócio da aplicação; É definido as funcionalidades do *software* e como os métodos se comportam. As classes da camada *Controller* são: PacienteController, FuncionarioController, EmpresaController, AvaliacaoPosturalController e AvaliacaoFisioterapeuticaController. Na Figura 12 são exibidas as classes existentes na camada Controller da aplicação, com os respectivos métodos.

[2] Disponível em: https://goo.gl/MrzE1K

Figura 12: Camada *Controller*.

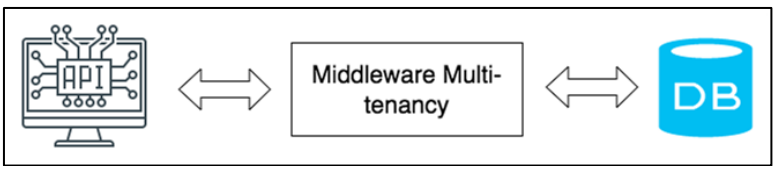


Fonte: Próprio Autor.

Por utilizar o *Laravel* para a implementação da API, não se faz necessário o uso da camada *Views* no projeto Laravel, responsável por exibir informações para o usuário, já que este papel será feito pela aplicação *front-end* desenvolvida utilizando o *framework Vue.js*.

Para que seja possível utilizar apenas uma instância do banco de dados e armazenar dados de diversas empresas, é necessário o uso do modelo *multi-tenancy*. Este modelo será responsável em filtrar todas as consultas por intermédio de um *middleware* a fim de garantir que os dados de uma empresa só serão disponibilizados para a mesma. Sempre que um dado for requisitado, o *multi-tenancy* verificará se esta informação pertence ao usuário que está requisitando e, caso positivo, devolverá a consulta ao usuário. Na Figura 13 é exibido um exemplo da como a arquitetura *multi-tenancy* funciona no consumo dos dados do banco.

Figura 13: Funcionamento do *multi-tenancy* na aplicação.



Fonte: Próprio Autor.

5.3 Implementação

O *software* desenvolvido consiste em duas aplicações, a primeira é a API RESTful, responsável por definir toda a lógica de negócio do sistema e a segunda é a aplicação *front-end* responsável por exibir toda interface para que o usuário possa manipular os dados utilizando a aplicação *web*. No desenvolvimento da API foi utilizado o *framework PHP Laravel*, que possui recursos que visam facilitar a implementação deste tipo de aplicação. Já no desenvolvimento da aplicação *web*, foi utilizado o *framework JavaScript Vue.js*, uma ferramenta utilizada para o desenvolvimento de aplicações *front-end* com comunicação *HTTP* entre servidores.

Nas próximas subseções será mostrado o processo de desenvolvimento das aplicações citadas, como elas se comunicam e o processo para disponibilizar o acesso à aplicação *front-end* por meio da Internet.

5.3.1 API RESTful

O objetivo da API desenvolvida é prover um caminho único para o acesso ao banco de dados, de forma que diferentes aplicações consigam compartilhar das mesmas informações. Por ser desenvolvida utilizando o *framework* Laravel, a API segue o padrão MVC. Apenas a camada V (*views*) não é utilizada, já que toda a parte *front-end* será desenvolvida utilizando outro *framework*.

A seguir será explicado o processo de desenvolvimento dos módulos de autenticação e *multi-tenancy*, além de como estes se integram à API. O código fonte completo do projeto pode ser acessado no repositório online[3].

5.3.1.1 Autenticação

A segurança dos dados é um importante passo para a construção de uma aplicação confiável, visto que deve-se garantir que os dados só sejam acessados por usuários devidamente cadastrados e autenticados no sistema. Sendo assim, a implementação da autenticação na API desenvolvida é a principal garantia da segurança desses dados.

Para a implementação da autenticação na aplicação, foi utilizado a biblioteca *Passport*. Essa biblioteca provê todos os recursos de autenticação via *token* utilizando o padrão OAuth2, utilizado mundialmente para a segurança de aplicações neste modelo.

[3] Disponível em: https://bitbucket.org/fagnerpsantos/dochelp_api

Segundo Andrey (2017), o OAuth 2 é um protocolo que permite aos usuários terem acesso limitado a recursos de *software* sem precisar expor suas credenciais. Normalmente, ao acessar uma aplicação que necessite de autenticação, um usuário precisa informar suas credenciais (login e senha). Após isso, o servidor verifica se estas informações são verdadeiras e disponibiliza acesso ao *software*. Porém, este tipo de autenticação abre algumas brechas para a segurança, já que a aplicação a ser utilizada possui acesso direto à senha do usuário.

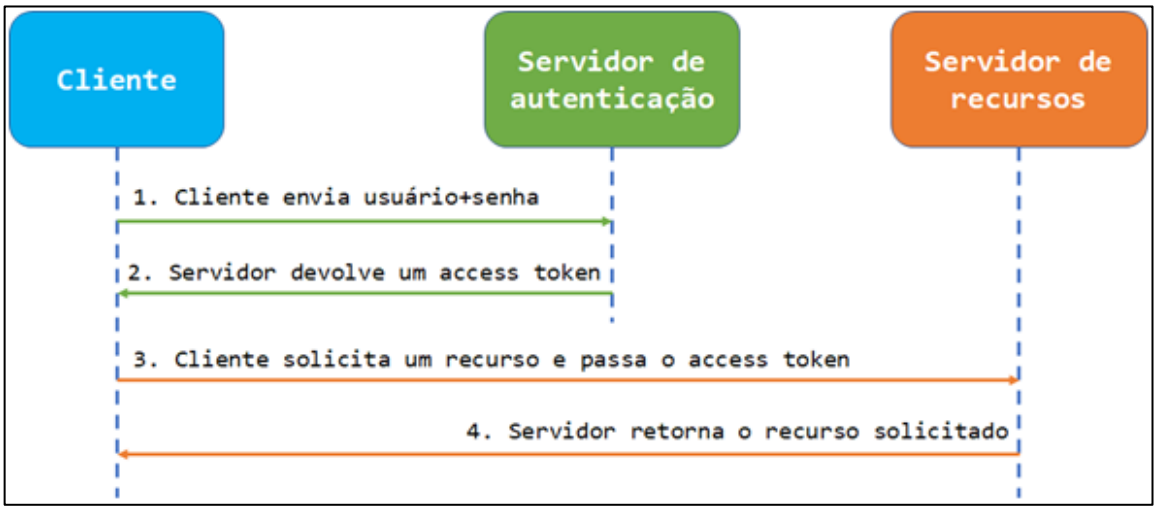
Sendo assim, segundo Faria (2015), o principal objetivo do OAuth 2 é permitir que uma aplicação se autentique em outra “em nome de um usuário” sem precisar ter acesso à sua senha.

Basicamente, segundo Andrey (2017), o fluxo básico do padrão OAuth 2 é o seguinte:

1. O usuário está em uma aplicação (cliente*)* e deseja acessar um recurso protegido do usuário em outra aplicação;
2. O cliente faz uma chamada ao servidor de autenticação por meio de uma rota passando 3 parâmetros (id do cliente, chave do cliente e rota de redirecionamento);
3. O usuário, então, fornece suas credenciais ao servidor de autenticação que permite o acesso aos recursos e devolve um *token* de acesso à aplicação. Este *token* é uma garantia de que o usuário que está acessando a aplicação foi permitido pelo servidor de autenticação;
4. O cliente, então, armazena este *token* para que todas as requisições feitas à API sejam liberadas.

O fluxo de autenticação utilizando a API pode ser vista na Figura 14. É evidente a importância da autenticação via *token* na API desenvolvida. Sempre que a aplicação *front-end* desejar acessar algum recurso do banco de dados, o usuário deverá passar suas credenciais que, posteriormente, serão validadas e retornadas o *token*. De posse deste *token*, a aplicação cliente poderá acessar os recursos do banco de dados da aplicação.

Figura 14: Fluxo básico da autenticação via *token.*



Fonte: Santos (2017).

5.3.1.2 Multi-tenancy

Uma aplicação *SaaS* é um *software,* no qual permite que os recursos de um servidor sejam compartilhados por diferentes empresas. Porém, um dos maiores problemas visto neste tipo de aplicação é a garantia do isolamento dos dados que, se não for feito corretamente, pode fazer com que diferentes usuários do sistema tenham acessos a dados de diferentes empresas, mesmo que estes não tenham permissão para tal.

Sendo assim, o uso do modelo *multi-tenancy* é muito importante no desenvolvimento dessas aplicações. Como foi discutido na Seção 2.3, o *multi-tenancy* permite que diferentes empresas utilizem um mesmo banco de dados. Porém, estes dados só poderão ser vistos por usuários permitidos. Por exemplo, os dados da empresa x só poderão ser vistos por funcionários que fazem parte da empresa x.

Visto isso, foi utilizada a biblioteca *Landlord*[4] para a implementação do *multi-tenancy* na API desenvolvida, disponível por meio da licença MIT (Instituto de Tecnologia de Massachusetts) de uso gratuito. Esta biblioteca tem como foco a utilização do *multi-tenancy* em aplicações que utilizem uma única instância do banco de dados para todos os clientes do *software*. Através da sua documentação é possível encontrar todas as informações para sua utilização, desde a instalação até o uso em aplicações *Laravel*.

[4] Disponível em: https://github.com/HipsterJazzbo/Landlord

Após a instalação e configuração da biblioteca, houve a necessidade da criação de um *middleware* no projeto. Um *middleware*, basicamente, é responsável por capturar a requisição que está sendo feita pelo usuário e tratá-la de forma definida pelo desenvolvedor. No desenvolvimento da API do projeto, o *middleware* trabalha como um robô verificando qual a empresa que o usuário logado faz parte e se o dado que este usuário está tentando acessar pertence a esta. Sendo assim, dentro do diretório app/Http/Middleware do projeto, foi criado um arquivo chamado AddEmpresaTenant.php. Dentro deste arquivo, define-se o método responsável por verificar a empresa do usuário e devolver apenas informações pertencentes a esta empresa. O código deste método pode ser visto na Figura 15.

Figura 15: Método responsável pelo *multi-tenancy* da aplicação.

```php
<?php

namespace App\Http\Middleware;

use Closure;

class AddEmpresaTenant
{
    /**
     * Handle an incoming request.
     *
     * @param  \Illuminate\Http\Request  $request
     * @param  \Closure  $next
     * @return mixed
     */
    public function handle($request, Closure $next)
    {
        if($request->is('api/*')){
            $user = \Auth::guard('api')->user();
            if($user){
                $empresa = $user->empresa;
                \Landlord::addTenant($empresa);
            }
        }
        return $next($request);
    }
}
```

Fonte: Próprio Autor.

Assim, garante-se que as informações de cada empresa só serão exibidas para o usuário que pertence a ela, o que faz com que o banco de dados da aplicação seja totalmente isolado.

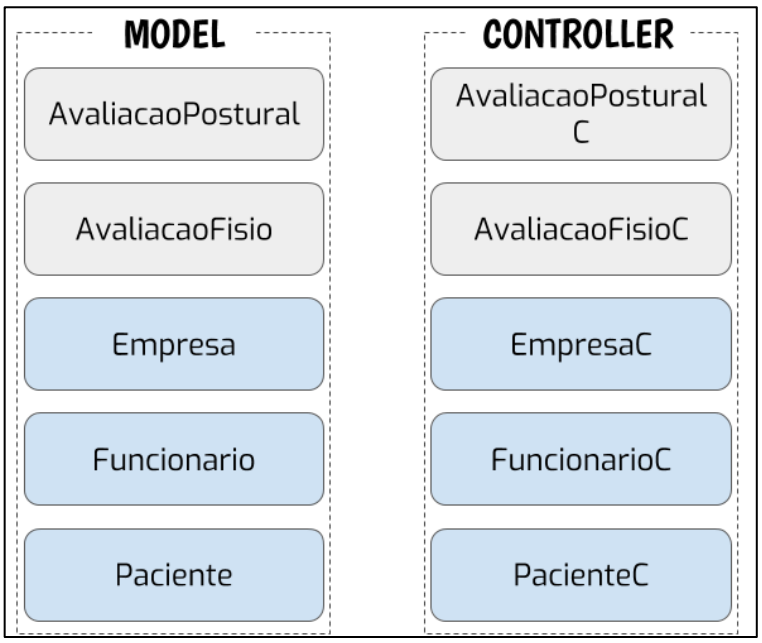
5.3.1.3 Implementação da API

O principal objetivo da API é prover um caminho único e central ao banco de dados, sendo que toda a lógica de negócio deve ser desenvolvida nesta aplicação, fazendo com que as aplicações clientes só precisem entender como as

requisições devem ser feitas (tipos de dados, rota, dentre outros). Assim, a API tem a responsabilidade de capturar as requisições feitas pelas aplicações cliente, verificar se estas requisições são válidas e responder estas requisições, seja o resultado de uma consulta ou o status de uma inserção no banco de dados.

Ao utilizar o Laravel como *framework* para o desenvolvimento da API, é necessário utilizar o padrão MVC para a implementação do projeto. Na Figura 16 é demonstrada a arquitetura geral do *back-end* da aplicação. Os *models* representam as entidades utilizadas no banco de dados e os *controllers* armazenam a lógica de negócio da aplicação.

Figura 16: Arquitetura da aplicação.



Fonte: Próprio Autor.

A camada *model* é responsável por mapear como cada entidade é formada no banco de dados e definir os relacionamentos que ocorrem entre elas (Figura 17).

Figura 17: Model da aplicação.

```php
<?php

namespace App;

use Illuminate\Notifications\Notifiable;
use Laravel\Passport\HasApiTokens;
use Illuminate\Foundation\Auth\User as Authenticatable;
use HipsterJazzbo\Landlord\BelongsToTenants;

class Funcionario extends Authenticatable
{
    use HasApiTokens, Notifiable, BelongsToTenants;

    protected $fillable = ['nome', 'email', 'password', 'cpf', 'rg', 'celular', 'cargo', 'estadoCivil',
    'dataNascimento', 'sexo', 'empresa_id'];

    protected $hidden = ['password', 'remember_token'];

    public function empresa(){
        return $this->belongsTo(Empresa::class);
    }
}
```

Fonte: Próprio Autor.

A camada *controller*, então, fica responsável por toda a lógica de negócio da aplicação; São implementados os métodos utilizados para a manipulação de cada entidade. Essa camada se comunica com a camada *model*, pois é ela quem tem a responsabilidade de comunicar com o banco de dados. Na Figura 18 pode ser visto um exemplo do *controller* e como ela funciona.

Figura 18: Método de listagem de todos os funcionários.

```php
public function index(Request $request)
{

        $qtd = $request['qtd'];
        $page = $request['page'];

        Paginator::currentPageResolver(function () use ($page) {
            return $page;
        });

        $funcionario = Funcionario::paginate($qtd);

        $funcionario = $funcionario->appends(Request::capture()->except('page'));

        return response()->json(['funcionarios'=>$funcionario], 200);

}
```

Fonte: Próprio Autor.

O método "index" utiliza o model Funcionario para capturar todos os registros da tabela funcionários do banco de dados e retornar, através de um JSON, as informações para o cliente.

O fluxo básico da API funciona da seguinte forma:

1. Aplicação cliente requisita um recurso, através de uma rota, para a API;
2. API processa essa requisição via *controller* e requisita ao model as informações necessárias;
3. O *model*, então, comunica com o banco de dados e devolve as informações para o *controller*;
4. O *controller*, então, devolve as informações requisitadas através do formato JSON como resposta à requisição realizada.

5.3.2 Aplicação *Front-end*

A aplicação *front-end* objetiva o consumo das informações providas através da API para que os usuários do sistema possuam uma interface amigável e simples de utilização do sistema. Esta aplicação foi desenvolvida utilizando o *framework* Vue.js, um *framework* de desenvolvimento JavaScript que provê todos os recursos necessários para a comunicação com o serviço *RESTful* entregue pela API.

A seguir será detalhado todo o processo de desenvolvimento da aplicação *front-end* e como esta se comunica e consome as informações vindas através do serviço RESTful desenvolvido. O código fonte do projeto completo pode ser acessado no repositório online[5].

5.3.2.1 Estrutura do Projeto

A aplicação segue a estrutura básica de um projeto Vue.js. Assim, todas as páginas são separadas em componentes, que possuem sua própria estrutura de HTML, CSS e JavaScript. O HTML é responsável por definir como a página será exibida para o usuário (botões, formulários, menus de navegação, dentre outros), já o CSS define o estilo da página HTML (cor dos botões, tamanho dos campos no formulário, dentre outros). Por fim, o JavaScript é responsável pelo comportamento da página (comunicação com o servidor, dados a serem exibidos na página HTML, dentre outros). Na Figura 19 pode ser visto um exemplo do código fonte de um componente utilizado no projeto.

Basicamente, um componente é dividido em três partes: HTML, JS e CSS. Cada parte possui uma *tag* que delimita seu início e fim e servem para indicar

[5] Disponível em: https://bitbucket.org/fagnerpsantos/dochelp_front

ao Vue.js o que deve ser executado em cada camada. Na *tag "template"* deve ser definido o HTML e a estrutura do componente. A *tag "script"* armazena o comportamento da página em código JavaScript. Por fim, a *tag "style"* possui a tarefa de agregar o código CSS que dará o visual ao HTML.

Figura 19: Componente de Login da aplicação.

```
<template>
<div class="hold-transition login-page" id="login">
  <div class="login-box">
  <div class="login-logo">…
  </div>
  <!-- /.login-logo -->
  <div class="login-box-body">…
  </div>
  <!-- /.login-box-body -->
</div>
</div>
</template>
```

HTML

```
<script>
import { CONFIG } from './../../config'
import Vue from 'vue'
export default {
    name: 'login',
    data: function () {…
    },
    methods: {…
    }
}
</script>
```

JS

```
<style>
#login{
    background-color: #eee !important;
    position: fixed;
    top: 0px;
    left: 0px;
    right: 0px;
    width: 100%;
    height: 100%;
    z-index: 99999999;
    padding: 20px;
}
</style>
```

CSS

Fonte: Próprio Autor.

5.3.2.2 Páginas do Sistema

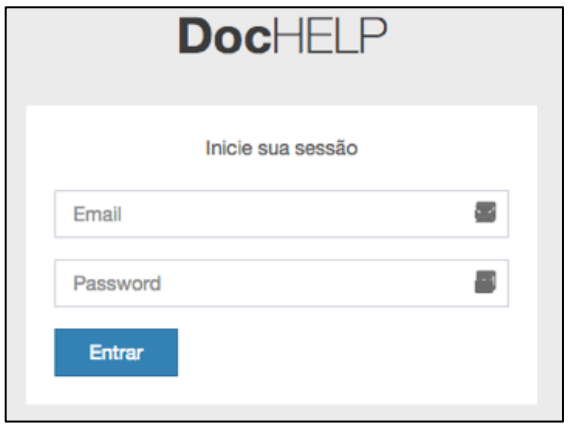
A aplicação conta com um fluxo de navegação bem definido. Para que seja possível acessar suas funcionalidades, o primeiro passo é cadastrar a empresa que utilizará o sistema. O cadastro pode ser feito utilizando o formulário de cadastro de empresas (Figura 20), no qual deve ser informado o nome, o e-mail e o CNPJ da empresa para que esta seja cadastrada no banco de dados do sistema.

Figura 20: Página de cadastro de empresas.

Fonte: Próprio Autor.

Após o cadastro realizado, um novo usuário será registrado no banco de dados que, por padrão, a senha criada para este usuário é “123456789” e o e-mail de acesso é o mesmo informado no momento da criação de uma empresa. Esses dados devem ser informados na tela de login da aplicação (Figura 21) para que o login seja feito e o usuário tenha acesso ao sistema e redirecionado à página principal (Figura 22), na qual todos os pacientes são exibidos.

Figura 21: Página de login.



Fonte: Próprio Autor.

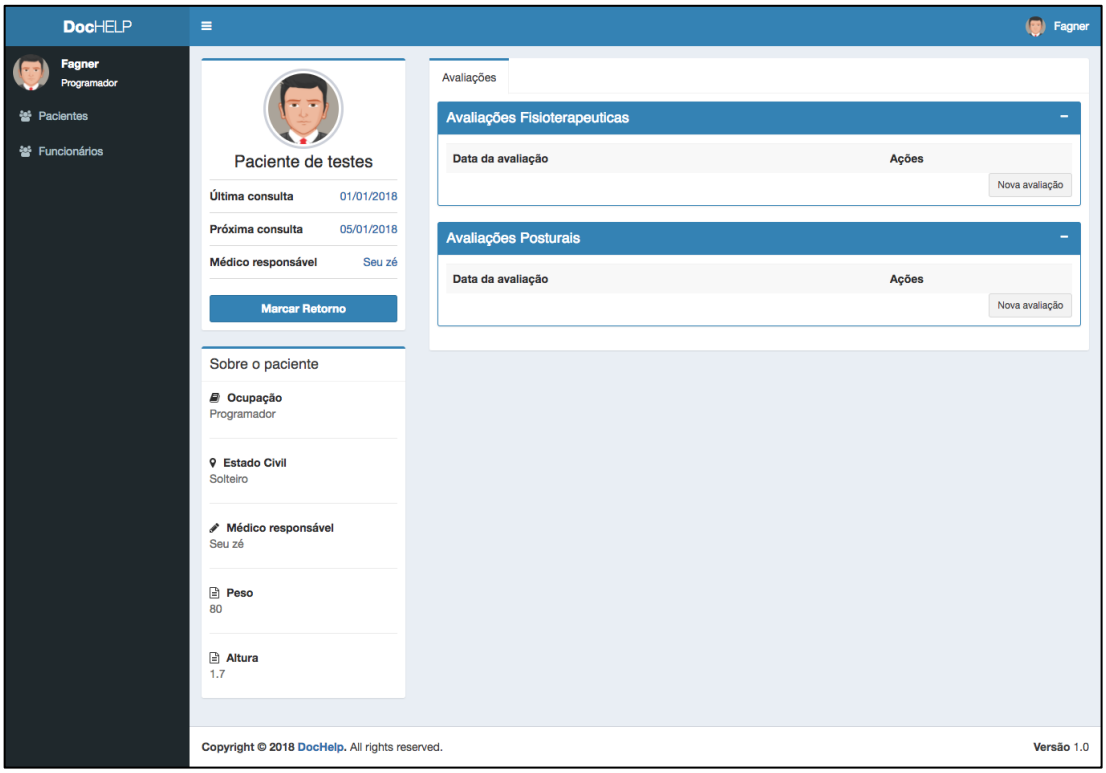
Na tela de listagem de pacientes (Figura 22), todos os pacientes cadastrados por aquela empresa serão listados e um botão para o cadastro de novos pacientes é exibido. Ao clicar sobre o nome do paciente, o usuário é redirecionado para a página do perfil do paciente (Figura 23), na qual todas as informações deste paciente são exibidas.

Figura 22: Página de listagem de pacientes.

Fonte: Próprio Autor.

Na tela de perfil do paciente (Figura 23) são exibidas todas as informações do paciente bem como as últimas avaliações do mesmo e um botão para criar uma nova avaliação.

Figura 23: Tela de perfil do paciente.



Fonte: Próprio Autor.

Por fim, na tela de listagem de funcionários (Figura 24) são mostrados os funcionários cadastrados por aquela empresa e um botão para que um novo funcionário seja criado no banco de dados.

Figura 24: Tela de listagem de funcionários.

Fonte: Próprio Autor.

5.3.2.3 Comunicação com a API

Para que o usuário da aplicação web consiga acessar os dados existentes no banco de dados, deve-se haver a comunicação entre a aplicação *front-end* e a API RESTful. A aplicação cliente solicita os recursos através da API e esta obtém os dados do banco de dados e os retorne para o cliente. Assim, cada componente possui um conjunto de métodos responsáveis por realizar esta comunicação. Basicamente, a requisição através da aplicação *web* ocorre da seguinte forma:

1. Cada requisição representa uma funcionalidade da aplicação. Esta requisição é denominada como uma *action*. Essa *action* é responsável por enviar uma requisição para a API e obter a resposta vinda através dessa requisição;
2. Após receber a resposta da requisição realizada, utiliza-se o método *commit* para atualizar as *mutations*, métodos responsáveis por alterar os estados que representam as coleções de dados;
3. Por fim, os estados representam as coleções de dados, nas quais são capturadas as informações a serem exibidas nas páginas da aplicação cliente.

Na Figura 25 pode ser visto como estes métodos funcionam e se relacionam, e como consequência provém a comunicação entre as aplicações.

Figura 25: Módulo de comunicação entre a aplicação *web* e a API.

```
import Vue from 'vue'
export default {
    state: {
        funcionarioList: []
    },

mutations: {
    updateFuncionarioList (state, data){
        state.funcionarioList = data
    }
},
actions: {
    getFuncionarios (context) {
        Vue.http.get('api/funcionarios').then(response => {
            context.commit('updateFuncionarioList', response.data)
        })
    },
    newFuncionario(context, data){
        Vue.http.post('api/funcionarios', data)
    }
}
}
```

Fonte: Próprio Autor.

Em seguida, é necessário determinar como serão realizadas as chamadas de cada método implementado no módulo anterior (Figura 25). Para isso, cada componente desenvolvido possui seus atributos e regras, nos quais são definidas as chamadas aos métodos de seu respectivo módulo (Figura 26).

Figura 26: Chamadas do componente para listar funcionários.

```
<script>
export default {
  name: 'ListarFuncionarios',
  methods: {
    listarFuncionario: function (id) {
      this.$router.push('/funcionarios')
    }
  },
  computed: {
    funcionarios () {
      return this.$store.state.funcionario.funcionarioList
    }
  },
  created () {
    this.$store.dispatch('getFuncionarios')
  }
}
</script>
```

Fonte: Próprio Autor.

O componente da Figura 26 é responsável por exibir todos os funcionários da empresa que o usuário autenticado trabalha. No atributo *methods*, são definidos os métodos utilizados pela página. No atributo *computed*, é capturado a lista de funcionários vinda por meio da API. Esta lista é definida no módulo de funcionários visto na Figura 25. Por fim, o método *created* indica qual método deve ser invocado ao renderizar o componente, listando todos os usuários.

5.3.3 *Deploy* da Aplicação

O processo de *deploy* de uma aplicação visa garantir a disponibilidade do *software* através da Internet, e que poderá ser acessado de qualquer parte do mundo por meio de um navegador de Internet.

Para que o *software* desenvolvido neste trabalho fosse acessado pelos clientes, garantindo que este seja considerado como um *SaaS*, o mesmo foi alocado em um servidor do serviço de nuvem chamado Heroku.

Segundo iMasters (2012), o Heroku é uma plataforma de serviço em nuvem que permite armazenar e disponibilizar aplicações por meio da Internet. Esta plataforma disponibiliza máquinas virtuais chamadas "*Dyno"*, que podem possuir desde configurações mais simples até grandes servidores de alto processamento.

O processo de envio e configuração de uma aplicação aos servidores do Heroku varia conforme a tecnologia utilizada. Por utilizar tecnologias *JavaScript* na aplicação cliente, foi necessário a utilização do *Express* para criar um servidor que possa acessar as pastas da aplicação, já que o Vue.js não possui tal recurso. Assim, é preciso instalar a dependência *express* no projeto utilizando o comando da Figura 27 .

Figura 27: Comando para instalar a dependência *express* no projeto.

```
npm install express --save
```

Fonte: Próprio Autor.

Após instalar a biblioteca *express* no projeto, é necessário realizar a configuração do servidor que executará a aplicação cliente por meio do *Heroku*. Para isso, é necessário realizar diversas configurações como visto na Figura 28.

Figura 28: Configurações do servidor *express*.

```
// server.js

var express = require('express');
var path = require('path');
var serveStatic = require('serve-static');

app = express();
app.use(serveStatic(__dirname + "/dist"));

var port = process.env.PORT || 5000;
app.listen(port);

console.log('server started '+ port);
```

Fonte: Próprio Autor.

Feito isso, ao acessar o endereço disponibilizado pelo *Heroku* para a aplicação, será possível visualizar a página inicial da aplicação cliente (Figura 21) e, com isso, a aplicação já pode ser acessada por qualquer usuário que possua um login válido no sistema. A aplicação pode ser acessada no repositório online[6].

5.4 Resultados

Após a implementação e o *deploy* do sistema nos servidores do *Heroku*, a aplicação foi testada para verificar se o modelo *multi-tenancy* proposto resolve os problemas levantados por este trabalho, a saber: segurança, integridade dos dados e redução de custos no desenvolvimento do *software*.

Após testar o *software* nas três clínicas de fisioterapia, foi analisado se o uso do modelo proposto possui benefícios que justifiquem sua utilização, além de avaliar se estes benefícios garantem a integridade, a segurança dos dados, e a diminuição dos custos de implantação e manutenção do *software*.

Em relação à garantia de integridade e segurança dos dados, pode-se comentar que, apesar dos clientes utilizarem o mesmo banco de dados, o *middleware,* responsável por garantir que os dados só serão acessados por seus respectivos proprietários, cumpre bem o seu papel, pois nos testes realizados não houve acesso indevido aos dados de uma empresa por funcionários de outra empresa. Portanto, o isolamento dos dados funciona de forma correta e garante a integridade e a segurança dos mesmos.

[6] Disponível em: https://dochelp-front.herokuapp.com

Sobre a redução de custos do projeto, é fácil perceber as melhorias que o uso deste modelo proporciona, já que, por utilizar um único banco de dados para diversos clientes, não há a necessidade do uso de um servidor dedicado para cada empresa do sistema. Além disso, por todo o processo de implantação e utilização ser feito por meio da Internet, não há a necessidade da compra de equipamentos por parte da empresa que pretende utilizar o *software*, o que garante uma economia muito grande quando comparado ao modelo de entrega “tradicional” de *software*.

6 CONCLUSÃO

O objetivo proposto por este trabalho foi de desenvolver uma aplicação SaaS utilizando o modelo *multi-tenancy* aplicado no contexto de computação em nuvem. Para tanto, primeiramente foi realizado um estudo sobre o tema, bem como o estado da arte no que diz respeito às tecnologias e métodos utilizados no uso dos sistemas *multi-tenancy*. Após isso, foi apresentado uma visão geral do modelo *multi-tenancy* e seus componentes, explicando seus conceitos e definições, além de apresentar as evidências sobre os fatores que influenciaram a adoção e desenvolvimento de aplicações SaaS em diversos segmentos.

Após isso, foi proposta a utilização do modelo *multi-tenancy* no desenvolvimento de *software* como serviço aplicado no contexto de computação em nuvem. Também foram abordadas as principais ferramentas e tecnologias para o desenvolvimento de uma aplicação deste tipo.

Além disso, foi desenvolvido um *software SaaS* multiplataforma utilizando os conceitos de *multi-tenancy* a fim de validar os benefícios deste modelo, além de implantar a aplicação em três clínicas fisioterapêuticas. Também foram coletados os resultados de melhorias no processo de gestão das clínicas.

Por fim, avaliou se o modelo utilizado supriu as necessidades e atendeu às demandas necessárias, o que garante a integridade e a segurança dos dados. Ademais, garante a redução dos custos operacionais na implantação e utilização do *software*. Assim, conclui-se que o modelo *multi-tenancy* atende bem às necessidades e garante a integridade dos dados mesmo com o uso de várias empresas em um único banco de dados.

6.1 Contribuições

Pode-se destacar como principais contribuições deste trabalho o desenvolvimento de um *software* como serviço utilizando o modelo *multi-tenancy* para gestão de clínicas fisioterapêuticas, sendo possível comprovar a eficiência desse modelo. Além disso, pode-se considerar de grande contribuição a descrição dos componentes utilizados para o desenvolvimento da aplicação bem como a comunicação entre si, o que auxiliará na criação de futuros *softwares* com este mesmo propósito.

6.2 Trabalhos Futuros

Como trabalhos futuros, há a possibilidade da utilização de testes unitários para auxiliar a implementação de novos *SaaS* utilizando o modelo descrito neste trabalho. Outra recomendação é a utilização de bancos de dados não-relacionais para análise de desempenho em relação ao uso dos bancos relacionais. Por fim, pode-se considerar a implementação de um aplicativo móvel como outra opção de cliente para consumo da API e, com isso, aumentar as possibilidades de utilização do modelo utilizado neste trabalho.

REFERÊNCIAS

ANDREY, L. **Como funciona o protocolo OAuth 2.0**. 2017. Disponível em: <https://imasters.com.br/desenvolvimento/como-funciona-o-protocolo-oauth-2-0/?trace=1519021197&source=single>. Acesso em: 8 out. 2017.

ASAAS, E. **Software como serviço:** Como tudo começou. 2013. Disponível em: <https://asaas.com/blog/*software*-como-servico-como-tudo-comecou/>. Acesso em: 20 out. 2017.

BARNES, F. **Os reais benefícios do ERP via SaaS**. 2014. Disponível em: <https://canaltech.com.br/gestao/Os-reais-beneficios-do-ERP-via-SaaS/>. Acesso em: 20 nov. 2017.

BELITSOFT. **React vs Angular vs Vue**. 2018. Disponível em: <https://belitsoft.com/front-end-development-services/react-vs-angular>. Acesso em: 26 mar. 2018.

BELLEI, E. A. **Sistema baseado em cloud computing para gestão de confinamento de gado de corte.** 2016. 62 f. Trabalho de Conclusão de Curso (Graduação) - Universidade Tecnológica Federal do Paraná, Pato Branco, 2016. Disponível em: <http://repositorio.roca.utfpr.edu.br/jspui/bitstream/1/7300/1/PB_COADS_2016_2_04.pdf>. Acesso em: 20 nov. 2017.

CHONG, F.; CARRARO, G. **Architecture Strategies for Catching the Long Tail**. 2006. Disponível em: <http://cistratech.com/whitepapers/MS_longtailsaas.pdf>. Acesso em: 10 dez. 2017.

COL, A. **Sistemas *multi-tenancy*:** desenvolvimento com Spring, HTML 5 e Bootstrap. 2015. 76 f. Trabalho de Conclusão de Curso (Especialização) - Universidade Tecnológica Federal do Paraná, Pato Branco, 2015. Disponível em: <http://repositorio.roca.utfpr.edu.br/jspui/bitstream/1/7181/1/PB_CEETJ_III_2015_01.pdf>. Acesso em: 20 nov. 2017.

CUSTÓDIO, A. F.; JUNIOR, E. A. O. **Um exemplo de aplicação *multi-tenancy* com Hibernate Shards**. 2012. 27 f. Trabalho de Conclusão de Curso (Especialização) - Universidade Estadual de Maringá, Maringá, 2012. Disponível em: <http://www.espweb.uem.br/site/files/tcc/2012/Anderson Fernando Custodio - Um exemplo de aplicacao *multi-tenancy* com Hibernate Shards.pdf>. Acesso em: 10 dez. 2017.

DALL'OGLIO, P. **PHP:** Programando com Orientação a Objetos. 3ª ed. São Paulo: Novatec, 2015.

DIAS, O. **Introdução ao Laravel**. 2014. Disponível em: <https://magazine.softerize.com.br/tutoriais/php/laravel/introducao-ao-laravel>. Acesso em: 5 jan. 2018.

ELMASRI, R.; NAVATHE, S. B. **Sistemas de Banco de Dados**. 4ª ed. São Paulo: Pearson Addison Wesley, 2005.

FARIA, A. **Papo Reto sobre OAuth 2**. 2015. Disponível em: <http://blog.andrefaria.com/papo-reto-sobre-oauth-2>. Acesso em: 10 out. 2017.

GIL, A. C. **Como Elaborar Projetos de Pesquisa.** 5ª ed. São Paulo: Atlas, 2010.

HARRIS, I. S.; AHMED, Z. An Open Multi-Tenant Architecture to Leverage SMEs. **European Journal of Scientific Research**, 2011.

IMASTERS. **Heroku.** 2012. Disponível em: <https://imasters.com.br/box/ferramenta/heroku/>. Acesso em: 02 abr. 2018.

KEARN, M. **Introduction to REST and .net Web API**. 2015. Disponível em: <https://blogs.msdn.microsoft.com/martinkearn/2015/01/05/introduction-to-rest-and-net-web-api/>. Acesso em: 20 jan. 2018.

MATERA, S. **O que é CSS e qual sua importância?.** 2012. Disponível em: <http://www.matera.com/br/2012/07/25/o-que-e-css-e-qual-sua-importancia/>. Acesso em: 6 jan. 2018.

MELL, P. M.; GRANCE, T. **The NIST definition of cloud computing**. 2011. Disponível em: <http://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-145.pdf>. Acesso em: 20 out. 2017.

MESQUITA, R. **Empresa SaaS:** a revolução dentro da sua empresa e seus benefícios. 2016. Disponível em: <https://saiadolugar.com.br/saas/>. Acesso em: 21 nov. 2017.

MUSPRATT, A. **Tapping into SaaS, PaaS and IaaS.** 2017. Disponível em: < https://blog.crozdesk.com/tapping-saas-paas-iaas/>. Acesso em: 20 mar. 2018.

NETO, J. R. **Desenvolvimento de aplicações *multi-tenancy*:** um estudo de mapeamento sistemático. 2012. 131 f. Dissertação (Mestrado) - Universidade Federal de Pernambuco, Recife, 2012. Disponível em: <http://200.17.137.109:8081/novobsi/Members/josino/material-extra/*multi-tenancy*/dissertacao corrigida_v2.pdf>. Acesso em: 10 nov. 2017.

OTERO, O. L. **Multi-tenant applications using Spring Boot, JPA, Hibernate and Postgres.** 2017. Disponível em: < http://tech.asimio.net/2017/01/17/Multitenant-applications-using-Spring-Boot-JPA-Hibernate-and-Postgres.html>. Acesso em: 20 mar. 2018.

PIRES, J. **O que é API? REST e RESTful? Conheça as definições e diferenças!** 2017. Disponível em: <https://becode.com.br/o-que-e-api-rest-e-restful/>. Acesso em: 21 jan. 2018.

PISA, P. **O que é e como usar o MySQL?.** 2012. Disponível em: <http://www.techtudo.com.br/artigos/noticia/2012/04/o-que-e-e-como-usar-o-mysql.html>. Acesso em: 5 jan. 2018.

REIS, V. **Por que VueJS é uma boa opção?.** 2016. Disponível em: <http://www.vuejs-brasil.com.br/por-que-vuejs-e-uma-boa-opcao/>. Acesso em: 4 jan. 2018.

RIBEIRO, M. **O que é API e como ele aumenta a produtividade nas empresas**. 2016. Disponível em: <https://pluga.co/blog/api/o-que-e-api/>. Acesso em: 21 jan. 2018.

SAASHOLIC. **Brazil SaaS Landscape 2017:** A Primeira Pesquisa Sobre o Mercado SaaS no Brasil. 2017. Disponível em: <https://saasholic.com/brasil-saas-landscape-2017-a-primeira-pesquisa-sobre-o-mercado-saas-no-brasil-5a6e6669cd6e>. Acesso em: 9 nov. 2017.

SANTOS, F. P. **Como Funciona o OAuth2**. 2017. Disponível em: <https://www.devmedia.com.br/view/viewaula.php?idcomp=38627>. Acesso em: 10 dez. 2017.

SILVA, S. M. **HTML 5:** A linguagem de marcação que revolucionou a Web. São Paulo: Novatec, 2014.

ZAIDMAN, A. Multi-Tenant SaaS Applications: Maintenance Dream or Nightmare? Position paper. **Iwpse-Evol ’10**, 2010.

APÊNDICE A – DESCRIÇÃO DOS CASOS DE USO

Quadro 8: UC-01 – Cadastrar Empresa.

Nome	UC-01 – Cadastrar Empresa
Atores	Usuário
Precondições	Nenhuma
Pós-condições	Uma nova empresa será cadastrada no sistema
Fluxo básico	**[1]** Este caso de uso se inicia quando uma empresa deseja realizar seu cadastro no sistema para ter acesso à aplicação. **[2]** O sistema exibe o formulário de cadastro com os campos nome, email e CNPJ, e os botões “Cadastrar”. **[3]** O usuário preenche o formulário com os dados da empresa. **[4]** O usuário clica no botão “Cadastrar”. **[5]** O sistema verifica se existe alguma empresa com o e-mail ou CNPJ informados no formulário. **[6]** O sistema grava as informações no banco de dados e exibe a página de login. **[7]** O caso de uso é encerrado.
Fluxos alternativos	**[2.1]** O usuário não preenche o formulário. [**2.2]** O caso de uso é encerrado. **[5.1]** O sistema encontra uma empresa com o mesmo e-mail ou CNPJ. **[5.2]** O sistema exibe uma mensagem de erro. **[5.3]** O caso de uso é encerrado.
Exceções	O sistema não consegue se comunicar com a API.

Fonte: Próprio Autor.

Quadro 9: UC-02 – Autenticar Usuário.

Nome	UC-02 – Autenticar Usuário
Atores	Usuário
Precondições	Devem existir usuários cadastrados no sistema
Pós-condições	O usuário será autenticado no sistema e, com isso, garantir acesso à aplicação.
Fluxo básico	**[1]** Este caso de uso se inicia quando um usuário deseja se autenticar no sistema. **[2]** O sistema exibe o formulário de login com os campos email e senha, e um botão “Entrar”. **[3]** O usuário preenche o formulário com os dados de login. **[4]** O usuário clica no botão “Entrar”. **[5]** O sistema verifica se existe algum usuário com os dados informados no formulário e autentica o usuário. **[6]** O sistema obtém o token de acesso através da API e grava na sessão. **[7]** O sistema exibe a tela inicial. **[8]** O caso de uso é encerrado.
Fluxos alternativos	**[3.1]** O usuário não preenche o formulário. [**3.2]** O caso de uso é encerrado. **[5.1]** O sistema não encontra um usuário com os dados informados. **[5.2]** O sistema exibe uma mensagem de erro. **[5.3]** O caso de uso é encerrado.
Exceções	O sistema não consegue se comunicar com a API.

Fonte: Próprio Autor.

Quadro 10: UC-03 – Cadastrar Funcionário.

Nome	UC-03 – Cadastrar Funcionário
Atores	Usuário
Precondições	O usuário deve estar autenticado no sistema.
Pós-condições	Um novo funcionário será criado no sistema.
Fluxo básico	**[1]** Este caso de uso se inicia quando um usuário deseja cadastrar um novo funcionário. **[2]** O sistema exibe a lista de usuários cadastrados na aplicação e um botão “Cadastrar”. **[3]** O usuário clica no botão “Cadastrar”. **[4]** O sistema exibe o formulário de cadastro de novos usuários. **[5]** O usuário preenche o formulário e clica em “Salvar”. **[6]** O sistema valida os dados do novo funcionário e salva no banco de dados. **[7]** O sistema exibe a lista de usuários cadastrados na aplicação. **[8]** O caso de uso é encerrado.
Fluxos alternativos	**[5.1]** O usuário não preenche o formulário. [**5.2]** O caso de uso é encerrado. **[6.1]** Os dados informados não são válidos. **[6.2]** O sistema exibe uma mensagem de erro. **[6.3]** O caso de uso é encerrado.
Exceções	O sistema não consegue se comunicar com a API.

Fonte: Próprio Autor.

Quadro 11: UC-04 – Cadastrar Paciente.

Nome	UC-04 – Cadastrar Paciente
Atores	Usuário
Precondições	O usuário deve estar autenticado no sistema.
Pós-condições	Um novo paciente será cadastrado no sistema.
Fluxo básico	**[1]** Este caso de uso se inicia quando um usuário deseja cadastrar um novo paciente. **[2]** O sistema exibe a lista de pacientes cadastrados na aplicação e um botão “Cadastrar”. **[3]** O usuário clica no botão “Cadastrar”. **[4]** O sistema exibe o formulário de cadastro de novos pacientes. **[5]** O usuário preenche o formulário e clica em “Salvar”. **[6]** O sistema valida os dados do novo paciente e salva no banco de dados. **[7]** O sistema exibe a lista de pacientes cadastrados na aplicação. **[8]** O caso de uso é encerrado.
Fluxos alternativos	**[5.1]** O usuário não preenche o formulário. [**5.2]** O caso de uso é encerrado. **[6.1]** Os dados informados não são válidos. **[6.2]** O sistema exibe uma mensagem de erro. **[6.3]** O caso de uso é encerrado.
Exceções	O sistema não consegue se comunicar com a API.

Fonte: Próprio Autor.

Quadro 12: UC-05 – Cadastrar Avaliação Postural.

Nome	UC-05 – Cadastrar Avaliação Postural
Atores	Usuário
Precondições	O usuário deve estar autenticado no sistema.
Pós-condições	Uma nova avaliação postural de um paciente será cadastrada no sistema.
Fluxo básico	**[1]** Este caso de uso se inicia quando um usuário deseja cadastrar uma avaliação postural de um paciente. **[2]** O sistema exibe a lista de pacientes cadastrados na aplicação. **[3]** O usuário seleciona o paciente que deseja cadastrar a avaliação postural. **[4]** O sistema exibe o perfil do paciente. **[5]** O usuário clica no botão “Nova avaliação postural” **[6]** O sistema exibe o formulário de cadastro de uma avaliação postural. **[7]** O usuário preenche o formulário. **[8]** O sistema valida os dados da avaliação e salva no banco de dados. **[9]** O sistema retorna para o perfil do paciente. **[10]** O caso de uso é encerrado.
Fluxos alternativos	**[7.1]** O usuário não preenche o formulário. [**7.2]** O caso de uso é encerrado. **[8.1]** Os dados informados não são válidos. **[8.2]** O sistema exibe uma mensagem de erro. **[8.3]** O caso de uso é encerrado.
Exceções	O sistema não consegue se comunicar com a API.

Fonte: Próprio Autor.

Quadro 13: UC-06 – Cadastrar Avaliação Fisioterapêutica.

Nome	UC-06 – Cadastrar Avaliação Fisioterapêutica
Atores	Usuário
Precondições	O usuário deve estar autenticado no sistema.
Pós-condições	Uma nova avaliação fisioterapêutica de um paciente será cadastrada no sistema.
Fluxo básico	**[1]** Este caso de uso se inicia quando um usuário deseja cadastrar uma avaliação fisioterapêutica de um paciente. **[2]** O sistema exibe a lista de pacientes cadastrados na aplicação. **[3]** O usuário seleciona o paciente que deseja cadastrar a avaliação fisioterapêutica. **[4]** O sistema exibe o perfil do paciente. **[5]** O usuário clica no botão “Nova avaliação fisioterapêutica” **[6]** O sistema exibe o formulário de cadastro de uma avaliação fisioterapêutica. **[7]** O usuário preenche o formulário. **[8]** O sistema valida os dados da avaliação e salva no banco de dados. **[9]** O sistema retorna para o perfil do paciente. **[10]** O caso de uso é encerrado.
Fluxos alternativos	**[7.1]** O usuário não preenche o formulário. [**7.2]** O caso de uso é encerrado. **[8.1]** Os dados informados não são válidos. **[8.2]** O sistema exibe uma mensagem de erro. **[8.3]** O caso de uso é encerrado.
Exceções	O sistema não consegue se comunicar com a API.

Fonte: Próprio Autor.