UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
DEPARTAMENTO ACADÊMICO DE INFORMÁTICA
CURSO DE ESPECIALIZAÇÃO EM TECNOLOGIA JAVA

EMERSON LUIS SIEGA

SISTEMA PARA REDES DE CLÍNICAS ODONTOLÓGICAS

MONOGRAFIA DE ESPECIALIZAÇÃO

PATO BRANCO
2015

EMERSON LUIS SIEGA

SISTEMA PARA REDES DE CLÍNICAS ODONTOLÓGICAS

Monografia apresentada na disciplina de Metodologia da Pesquisa, do Curso de Especialização em Tecnologia Java, do Departamento Acadêmico de Informática, da Universidade Tecnológica Federal do Paraná, Câmpus Pato Branco, como requisito parcial para obtenção do título de Especialista.

Orientadora: Profa. Beatriz Terezinha Borsoi

PATO BRANCO
2015

SISTEMA PARA REDES DE CLÍNICAS ODONTOLÓGICA

Por

Emerson Luis Siega

Esta monografia foi apresentada às 17h30 do dia 27 de outubro de 2015 como requisito parcial para a obtenção do título de ESPECIALISTA, no III Curso de Especialização em Tecnologia Java, da Universidade Tecnológica Federal do Paraná, Câmpus Pato Branco. O acadêmico foi arguido pela Banca Examinadora composto pelos professores abaixo assinados. Após deliberação, a Banca Examinadora considerou o trabalho aprovado.

Profa. Dra. Beatriz Borsoi
Orientadora
UTFPR – Campus Pato Branco

Prof. Msc. Rubia Eliza de Oliveira Schutz Ascari
Banca
UTFPR – Campus Pato Branco

Prof. Msc. Robison Cris Brito
Banca
UTFPR – Campus Pato Branco

Prof. Msc. Robison Cris Brito
Coordenador do curso de Especialização
UTFPR – Campus Pato Branco

RESUMO

SIEGA, Emerson Luis. Sistema para redes de clínicas odontológicas. 2015. 42 f. Monografia (Trabalho de especialização) – Departamento Acadêmico de Informática, Universidade Tecnológica Federal do Paraná, Câmpus Pato Branco. Pato Branco, 2015.

Este trabalho apresenta a modelagem e o desenvolvimento de uma aplicação *web,* que contém os módulos básicos do que se tornará um sistema de grande porte e completo para o gerenciamento de redes de consultórios odontológicos. O objetivo desse sistema, quando completo, será o de proporcionar o controle de todos os setores e procedimentos existentes dentro de uma clínica odontológica, permitindo também o controle agrupado das clínicas de um mesmo grupo (franquias). A necessidade para o desenvolvimento de uma solução como essa foi identificada por meio de conversas informais e visitas a dentistas da região Sudoeste do Paraná. Constatou-se que muitas clínicas ainda não utilizam um sistema para realizar agendamento, controle das consultas e cobranças, e as que utilizam esse tipo de sistema, não estavam satisfeitas com as funcionalidades e/ou preços disponibilizados por eles. A partir disso, por meio de um interessado no desenvolvimento do sistema, foi feita uma análise dos principais módulos e recursos que um sistema com essa finalidade deve conter. Tendo elencados esses recursos, foram selecionados alguns módulos básicos para serem desenvolvidos no presente trabalho. Foram implementados então os módulos de autenticação e autorização de usuários, os quais foram desenvolvidos como um projeto, que pode ser integrado a qualquer outro sistema. Foi desenvolvido, ainda, um módulo de controle de pacientes, clínicas e organizações, bem como dos usuários do sistema relacionados às clínicas. A aplicação atende conceitos de computação em nuvem e foi desenvolvida com a linguagem Java e utilizando tecnologias como AngularJS e Bootstrap. Openshift foi utilizada como plataforma para hospedagem e o Wildfly como servidor da aplicação. O banco de dados utilizado é o PostgreSQL

Palavras-chave: Redes de clínicas. Computação em nuvem. Consultório odontológico.

ABSTRACT

SIEGA, Emerson Luis. Software for network of dental clinics. 2015. 42 f. Monografia (Trabalho de especialização) – Departamento Acadêmico de Informática, Universidade Tecnológica Federal do Paraná, Câmpus Pato Branco. Pato Branco, 2015.

This work presents the modeling and development of a web application, which contains the basic modules of what will become a large scale and complete system for total management of network of dental clinics. The goal of this system, when complete, will be provide a control of all the sectors and procedures within a dental clinic, also allowing a clustered control of clinics of the same group (franchises). The necessity to develop such a solution was identified through informal conversations and visits to dentists of Southwest Paraná. It was identified that many clinics still do not use a system to perform scheduling, control of appointment and payments, and those that already use a system, were not satisfied with the features and/or prices offered by them. From there, through an interested in developing the system, an analysis was made of the main modules and features that a system for this purpose should include. Having listed these features, some basic modules were selected to be developed in this work. Then the authentication and user authorization modules was created, which were implemented in a separate project, which can be integrated with any other system. The developed features are: patient control module, control of clinics and organizations, and system users related to clinical. The software is cloud computing and was develop in Java language with AngularJS and Bootstrap. Openshift is the platform for hosting the application and Wildfly is the application server. PosgreSQL is the database.

Keywords: Network clinics. Cloud computing. Dental clinics.

LISTA DE FIGURAS

LISTA DE QUADROS

LISTAGENS DE CÓDIGOS

LISTA DE SIGLAS

CSS	*Cascading Style Sheets*
HTTP	*Hypertext Transfer Protocol*
laaS	*Infrastructure as a Service*
JDK	*Java Development Kit*
MVC	*Model–View–Controller*
NIST	National Institute of Standards and Technology
PaaS	*Platform as a Service*
SaaS	*Software as a Service*
TI	Tecnologia de Informação
UML	*Unified Modeling Language*
URL	*Uniform Resource Locator*
VPN	*Virtual Private Network*
XaaS	*Anything as a Service*

SUMÁRIO

1 INTRODUÇÃO

Este capítulo apresenta as considerações iniciais do trabalho com o contexto da aplicação desenvolvida, os seus objetivos e a justificativa. O texto é finalizado com a apresentação dos capítulos subsequentes.

1.1 CONSIDERAÇÕES INICIAIS

A essência da computação em nuvem está na possibilidade de acessar arquivos e executar tarefas pela Internet, sendo que esses arquivos e serviços estão armazenados em servidores que tem a finalidade de provê-los. O acesso aos serviços, como funcionalidades de *software*, por exemplo, providas por servidores e repositórios de dados é realizado mediante pagamento específico ou de maneira gratuita. E eles são, comumente, providos por instituições especializadas no oferecimento desses serviços, mas, geralmente, não diretamente relacionadas à produção dos serviços oferecidos.

Os serviços providos pela computação em nuvem desvinculam as empresas desenvolvedoras de *software* de prover o armazenamento dos aplicativos que são utilizados pelos seus clientes, por exemplo. Essa funcionalidade é delegada a uma empresa especializada que se encarrega de fornecer o acesso e manter os dados armazenados.

Esse tipo de serviço pode ser bastante útil quando uma rede de empresas utiliza um mesmo sistema, como, por exemplo, redes de consultórios médicos e/ou odontológicos, de supermercados e de lojas de departamentos. Uma instituição especializada provê o acesso ao aplicativo e o armazenamento e a disponibilização dos dados. Ocorre, assim, uma espécie de terceirização da disponibilização da aplicação e do armazenamento dos dados.

O sistema desenvolvido como resultado deste trabalho se insere no contexto de disponibilização de serviço, que é respaldado pelo conceito de computação em nuvem. O sistema se destina ao gerenciamento de redes de consultórios odontológicos. O sistema é composto por diversos módulos compreendendo gerenciamento da organização (rede de clínicas), de pacientes e de usuários. Os recursos oferecidos pela computação em nuvem facilitam o acesso e o gerenciamento dos dados entre as diversas clínicas que estão geograficamente distribuídas.

1.2 OBJETIVOS

A seguir são apresentados os objetivos da realização deste trabalho.

1.2.1 Objetivo Geral

Implementar funcionalidades que comporão a base de um sistema para gerenciamento de redes de clínicas odontológicas.

1.2.2 Objetivos Específicos

Facilitar o compartilhamento de dados entre clínicas distintas pelo uso de conceitos de computação em nuvem.

Prover uma interface simples e personalizável, que possa ser utilizada como ferramenta de trabalho no dia-a-dia de profissionais da área de odontologia.

Possibilitar o controle de acesso a cada módulo do sistema.

1.3 JUSTIFICATIVA

O uso de conceitos de computação em nuvem facilita o acesso e o compartilhamento dos dados, especialmente, quando esses dados provêm de origens distintas, como é o caso de empresas que possuem filiais ou são distribuídas.

A computação em nuvem permite a centralização dos dados e dos aplicativos sem que a própria empresa se preocupe em manter os servidores de dados e de aplicação. Esses serviços são terceirizados para empresas especializadas que se comprometem, inclusive, com a segurança dos dados.

O desenvolvimento de um sistema para o gerenciamento de um grupo de clínicas odontológicas facilita o compartilhamento dos dados entre as clínicas e o gerenciamento da segurança desses dados. Esses aspectos não estão relacionados à nuvem em si, um servidor *web* mantido pela própria empresa teria a mesma funcionalidade. O aspecto ressaltante é que esses serviços estarão sendo realizados por uma empresa especializada e, assim, as clínicas podem dedicar-se ao seu negócio.

1.4 ESTRUTURA DO TRABALHO

O texto está estruturado em capítulos. Este apresentou a introdução com o contexto do trabalho, os seus objetivos e a justificativa. O Capítulo 2 apresenta o referencial teórico que é sobre computação em nuvem. No Capítulo 3 são apresentadas as ferramentas e as tecnologias utilizadas na modelagem e na implementação do trabalho. Os resultados obtidos com a realização do trabalho são apresentados no Capítulo 4. E as considerações finais estão no Capítulo 5, seguidas das referências utilizadas para fundamentar o trabalho.

2 REFERENCIAL TEÓRICO

O referencial teórico apresenta a fundamentação conceitual do trabalho que se refere à computação em nuvem.

2.1 CLOUD COMPUTING

O termo "*cloud*", ou seja, nuvem, originou-se das telecomunicações quando provedores começaram a usar serviços de redes virtuais privadas, as *Virtual Private Network* (VPN) para a comunicação de dados (HARAUZ; KAUFINAN; POTTER, 2009).

O National Institute of Standards and Technology (NIST) define computação em nuvem, em inglês *cloud computing*, como sendo um modelo para permitir acesso à rede de forma conveniente e sob demanda a um *pool* de recursos de computação configuráveis (como, por exemplo, redes, servidores de aplicações e serviços compartilhados) que podem ser facilmente fornecidos e liberados com um esforço mínimo de gerenciamento ou interação com o provedor do serviço (NATIONAL..., 2015).

Computação em nuvem lida com computação, software, acesso a dados, armazenamento de serviços que podem não requerer o conhecimento do usuário final para a localização física e a configuração do sistema que está entregando o serviço (JADEJA; MODI, 2012). E tem se apresentado como a tendência recente em Tecnologia de Informação (TI) que move computação e dados dos *desktops* e computadores portáteis para grandes *data centers* (DIKAIAKOS *et al*., 2009).

Computação em nuvem é considerada a próxima evolução do paradigma de computação distribuída que provê recursos de armazenamento e computação (BEHL; BEHL, 2012). Para Shetty (2013) a computação em nuvem é claramente uma das tecnologias mais atraentes devido a sua escalabilidade, flexibilidade e efetividade em termos de custo para acessar recursos computacionais. Contudo, enquanto para alguns, essa é apenas uma forma simples de acessar software e armazenar dados na nuvem, para outros é um valor agregado para os processos existentes, definindo como podem levar vantagem desse modelo distribuído (CLOUD..., 2015).

Com a proliferação em larga escala da Internet, aplicações podem ser entregues como serviços sobre a Internet e isso reduz os custos (JADEJA; MODI, 2012). Para esses autores o

objetivo principal da computação em nuvem é fazer o melhor uso dos recursos distribuídos, combinando-os para obter a melhor taxa de transferência e ser capaz de resolver problemas de computação que exigem muito processamento. E, ainda, que a computação em nuvem lida com virtualização, escalabilidade, interoperabilidade, qualidade de serviço e os modelos de entrega da nuvem, ou seja, privada, pública e híbrida.

O desempenho da maioria das aplicações que usam o ambiente de computação em nuvem depende crucialmente da infraestrutura de rede entre o provedor da nuvem e os seus usuários (SHETTY, 2013). Para Yamini (2012) computação em nuvem pode ser classificada como um novo paradigma para provimento de serviços dinâmicos de computação suportados pelo estado da arte em *data centers* que geralmente empregam tecnologias de máquinas virtuais com propósitos de isolamento de ambiente.

A computação em nuvem é o resultado de diversos fatores como a tecnologia de computação tradicional e comunicação e o modelo de negócio. E é baseada na rede e tem o formato de serviços para o consumidor (LIU, 2012). De acordo com esses autores, o sistema de computação em nuvem provê o serviço para o usuário e possui a característica de alta escalabilidade e confiabilidade.

Diversas empresas provêem plataformas para computação em nuvem, ou mais especificamente, a hospedagem. Dentre essas empresas estão (LIU, 2012): Google, IBM, Microsoft, Amazon, VMware e EMC.

A visão de utilitários de computação baseados em modelo de provisão de serviços antecipa a massiva transformação da indústria da computação no século 21 que envolve a rapidez de disponibilização de serviços sob demanda (YAMINI, 2012). Com a computação em nuvem, os usuários (consumidores) pagam somente pelos serviços utilizados sem preocupar-se com a infraestrutura de armazenamento ou mesmo a localização desses serviços.

A computação em nuvem fornece infraestrutura, plataforma e software (aplicações) como serviços, que são disponibilizados para os usuários como assinatura baseada em serviços de acordo com um modelo de pagamento que é calculado pelo uso do serviço. Na indústria esses serviços são conhecidos como (SHAIKH; HAIDER, 2011, YAMINI, 2012, AMANATULLAH *et al.,* 2013):

a) infraestrutura orientada a serviços (*Infrastructure as a Service* (laaS)) – é permitido aos usuários utilizar os serviços que estão hospedados no servidor da nuvem;

b) plataforma como serviço (*Platform as a Service* (PaaS)) – os clientes podem armazenar seus próprios aplicativos na nuvem;

c) software como serviço (*Software as a Service* (SaaS)) – aluguel de processamento, armazenamento de dados e outros recursos básicos de computação são concedidos, permitindo aos usuários gerenciar sistemas operacionais, aplicações, armazenamento e conectividade de rede;

d) qualquer coisa como serviço (*Anything as a Service* (XaaS)) – os consumidores que possuem as suas próprias definições de serviços e requisitos para suas soluções de negócio e esperam que um provedor de nuvem possa gerenciar e atender aos seus requisitos.

3 MATERIAIS E MÉTODO

Este capítulo apresenta as ferramentas e as tecnologias e o método utilizados para a modelagem e a implementação do sistema.

3.1 MATERIAIS

O Quadro 1 apresenta as ferramentas e as tecnologias utilizadas para a modelagem e a implementação do sistema.

Astah Communitty	6.9	http://astah.net/editions/community	Modelagem Un*nified Modeling Language (*UML)
SAP Sybase PowerDesigner	15.3	http://www.powerdesigner.de/en	Modelagem do banco de dados
Java Platform (Java Development Kit)	7u71	http://www.oracle.com/	Linguagem de Programação
Eclipse Luna	4.4.2	https://www.eclipse.org/	Ambiente de desenvolvimento
AngularJS	1.3	https://angularjs.org/	Framework *Model–View–Controller* (MVC) para *front-end*
Bootstrap	3.3	http://getbootstrap.com/	Biblioteca de *Cascading Style Sheets* (CSS) e JavaScript
PostgreSQL	9.4.1	http://www.postgresql.org/	Banco de dados
Wildfly	8.2	http://wildfly.org/	Servidor de aplicação
OpenShift		https://www.openshift.com/	Plataforma *cloud* para hospedagem da aplicação

Quadro 1 – Ferramentas e tecnologias utilizadas

3.2 MÉTODO

Como método para realizar as atividades relacionadas à modelagem e à implementação do sistema foi utilizado o modelo iterativo e incremental (KRUCHTEN, 1995).

A necessidade de um sistema para gerenciamento de redes de consultórios odontológicos foi identificada a partir da conversas informais com dentistas da região Sudoeste do Paraná. Um dos profissionais relatou que já utilizou três sistemas distintos, mas

nenhum deles atendia as suas reais necessidades de negócio. Esse profissional foi o fornecedor dos requisitos para as iterações já realizadas e o será para as próximas iterações desenvolvidas. E ele também será o cliente piloto da aplicação.

À medida que os requisitos foram identificados, eles foram modelados e implementados. O levantamento dos requisitos foi realizado pelas funcionalidades que o cliente identificava para o sistema. Essas funcionalidades, uma vez identificadas eram documentadas e vinculadas aos respectivos módulos. A implementação foi e será realizada por módulos, como descritos na Seção 4.1. Esses módulos são complementados em várias iterações.

Para a implementação foram utilizadas as tecnologias descritas no Quadro 1 e testes de código e de usabilidade foram realizados paralelamente ao desenvolvimento do sistema. O cliente que forneceu os requisitos realizou testes de funcionalidades, visando identificar se os requisitos pretendidos por ele estavam sendo atendidos.

4 RESULTADOS

Neste capítulo são apresentados os resultados da realização do trabalho. Esses resultados se referem à apresentação do sistema, descrição e modelagem das funcionalidades e exemplos da codificação realizada.

4.1 ESCOPO DO SISTEMA

Em um primeiro momento foi desenvolvida a base inicial de um sistema para gerenciamento de redes de consultórios odontológicos. Como resultado da realização deste trabalho apenas os módulos básicos foram desenvolvidos, pois um sistema que realiza o gerenciamento completo de um consultório odontológico é um sistema de grande porte e complexo.

O objetivo deste sistema, quando implementadas todas as funcionalidades consideradas necessárias, é fornecer os meios para uma rede de clínicas gerenciar os consultórios individualmente e como grupos. A meta nesse trabalho é fornecer uma base inicial bem estruturada com recursos básicos, para futuramente receber de forma gradativa os demais módulos e recursos definidos para o sistema.

Os recursos a serem desenvolvidos são: gerenciamento da organização (rede de clínicas), de pacientes e de colaboradores usuários do sistema (recepcionistas, dentistas, administradores, entre outros). Além desses recursos, também haverá um controle de autenticação e autorização, que é essencial para qualquer sistema. Esses módulos são detalhados na sequência. Embora somente os três primeiros módulos tenham sido implementados, o módulo de gerenciamento de consultas é apresentado para fornecer uma visão geral do produto, quando implementadas todas as suas funcionalidades.

a) Gerenciamento da organização

Nesse módulo será possível realizar todas as configurações dos consultórios, como as informações básicas de cadastro, algumas configurações de aparência do sistema e informações visuais, como logotipos.

b) Gerenciamento de pacientes

Esse módulo será responsável pelo gerenciamento das informações cadastrais dos pacientes. Nesse conjunto de informações cadastrais estarão presentes: dados principais,

endereço, contato, dados clínicos e de histórico de atendimentos.

c) Gerenciamento de colaboradores

Esse será o módulo que possibilitará o gerenciamento dos funcionários da empresa. Os funcionários serão divididos por empresa e terão informações de cadastro diferenciadas de acordo com sua ocupação. Dentro do sistema, os colaboradores serão tratados como usuários do sistema, podendo ser classificados como recepcionistas, dentistas, entre outros.

d) Gerenciamento de consultas

Esse módulo disponibilizará a agenda de consultas e a visualização dos atendimentos que já foram realizados, que estão sendo realizados no momento e os agendados para datas futuras. A agenda será bem completa e ao mesmo tempo de uso simples e deve ser individualizada, ou seja, cada profissional terá acesso a sua agenda. A agenda possibilitará, também, o pré-cadastramento de pacientes no ato do agendamento da consulta.

Alguns detalhes técnicos relevantes sobre o sistema:

a) Será disponibilizado em ambiente *cloud*;

b) Será utilizado apenas um banco de dados para todas as empresas, separando suas informações por *schemas*;

c) O controle de autenticação será feito por meio de um projeto separado, possibilitando futuro reaproveitamento da lógica;

d) O *front-end* será desenvolvido em um projeto a parte e toda a comunicação com o *back-end* será feita via serviços REST.

4.2 MODELAGEM DO SISTEMA

No Quadro 2 é apresentada a listagem dos requisitos funcionais do sistema. Os requisitos RF01, RF02 e RF03 foram desenvolvidos como trabalho de conclusão de curso. Os demais serão implementados futuramente, de forma gradativa.

Identificação	Nome	Descrição
RF01	Gerenciar clínicas e organizações	Possibilitar o cadastramento e gerenciamento das clínicas que compõem uma organização (franquias), ou clínicas individuais. Contendo informações básicas das clínicas, endereço, contato, entre outros.
RF02	Gerenciar pacientes	Possibilitar o cadastramento e gerenciamento de pacientes. Serão mantidas informações básicas, endereço, contato, histórico odontológico, histórico de atendimentos (Consultas).

RF03	Gerenciar colaboradores	Possibilitar o cadastramento de usuários do sistema. Usuários do sistema são os próprios colaboradores da clínica, que podem ser desde recepcionistas até os próprios dentistas.
RF04	Gerenciar consultas	Possibilitar que sejam marcadas e agendadas consultas com todos os dentistas da clínica. Deve ser possível obter uma visualização em tempo real do fluxo de atendimentos da clínica (agendamentos do dia, pacientes na sala de espera, pacientes em atendimento, consultas remarcadas). Também deve estar disponível para cada usuário dentista a sua agenda de atendimentos.
RF05	Gerenciar estoque de produtos	Possibilitar o cadastramento e gerenciamento dos produtos utilizados nas clínicas.
RF06	Gerenciar fornecedores	Manter informações sobre os fornecedores de cada produto utilizado.
RF07	Contas a receber	Possibilitar o gerenciamento completo das contas a receber.
RF08	Contas a pagar	Possibilitar o gerenciamento completo das contas a pagar.
RF09	Notificar consultas por SMS	Possibilitar o envio de mensagens de texto para lembrar os pacientes de suas consultas e solicitar confirmação de presença.
RF10	Emissão de boletos	Possibilitar a emissão de boletos após aprovação de orçamentos e procedimentos com pacientes.

Quadro 2 – Requisitos funcionais

O Quadro 3 apresenta a listagem de alguns dos requisitos não funcionais levantados para o desenvolvimento do sistema. Esses requisitos especificam questões como segurança, desempenho, qualidade, entre outros.

Identificação	Nome	Descrição
RNF01	Acesso ao sistema	O acesso ao sistema será realizado por meio de e-mail e senha, onde o usuário terá acesso a um *token* e a partir deste *token*, irá acessar todos os módulos disponíveis a ele.
RNF02	Permissões de acesso	Cada usuário só deve acessar os módulos/menus que estiverem disponíveis nas suas permissões de acesso.
RNF03	Disponibilização como serviço	Os usuários do sistema não devem se preocupar com compra de máquinas e servidores. O sistema deve ser disponibilizado como serviço e seu acesso ocorrerá via navegador *web*.
RNF04	Alta disponibilidade	Mesmo não estando em servidores locais, o sistema deve apresentar uma alta disponibilidade, para não afetar o trabalho da clínica.

Quadro 3 – Requisitos não funcionais

As próximas duas Figuras apresentam os diagramas de entidades e relacionamentos de sistemas desenvolvidos no presente trabalho.

O primeiro diagrama, apresentado na Figura 1, representa o banco de dados do sistema de autenticação. Esse é um sistema independente, desenvolvido para poder ser integrado com qualquer outro sistema. É o responsável pelo gerenciamento de *tokens* de acesso para que usuários possam acessar aplicações cadastradas em seu banco de dados.

Além da autenticação, também é possível cadastrar no banco de dados as regras de autorização de usuários para cada aplicação. O controle de autorização será feito posteriormente pelos sistemas integrados a essa aplicação.

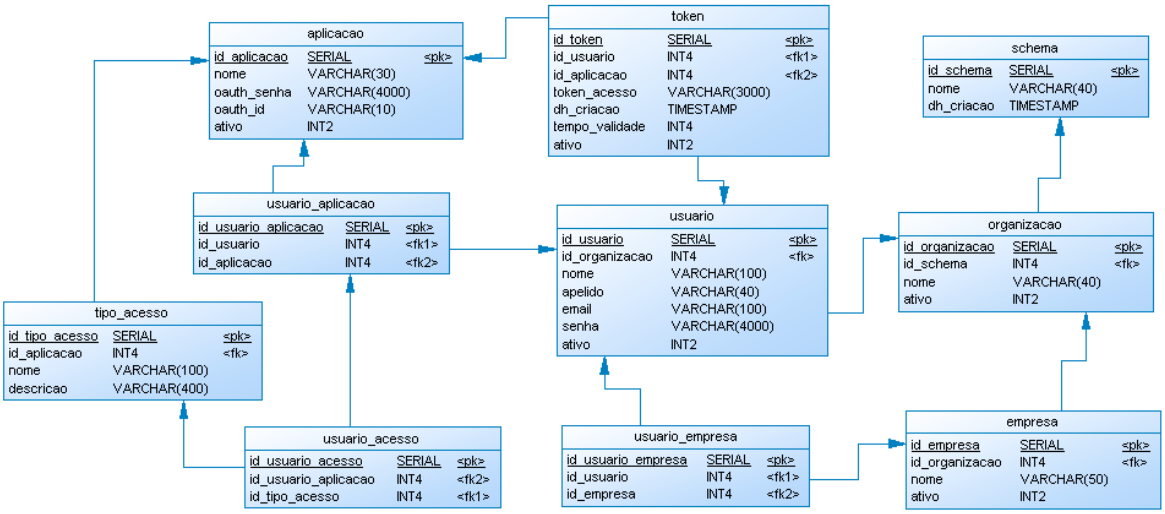

Figura 1 – Diagrama de entidades e relacionamentos do banco de dados do servidor de autenticação

O diagrama apresentado na Figura 2 é o de entidades e relacionamentos do próprio sistema para gerenciamento de clínicas odontológicas. Este diagrama está separado em dois *schemas*, um deles possui tabelas com dados compartilhados entre todas as organizações que utilizam o sistema, chamado "*public*". O outro *schema* representa tabelas com dados específicos de uma organização (conjunto de clínicas ou uma única clínica). Para cada organização que utilizará o sistema, será criado um novo *schema* específico com seu nome que conterá todas as suas informações. Desta forma, mantém-se separados os dados de clínicas diferentes, não permitindo que uma clínica acesse as informações de outra.

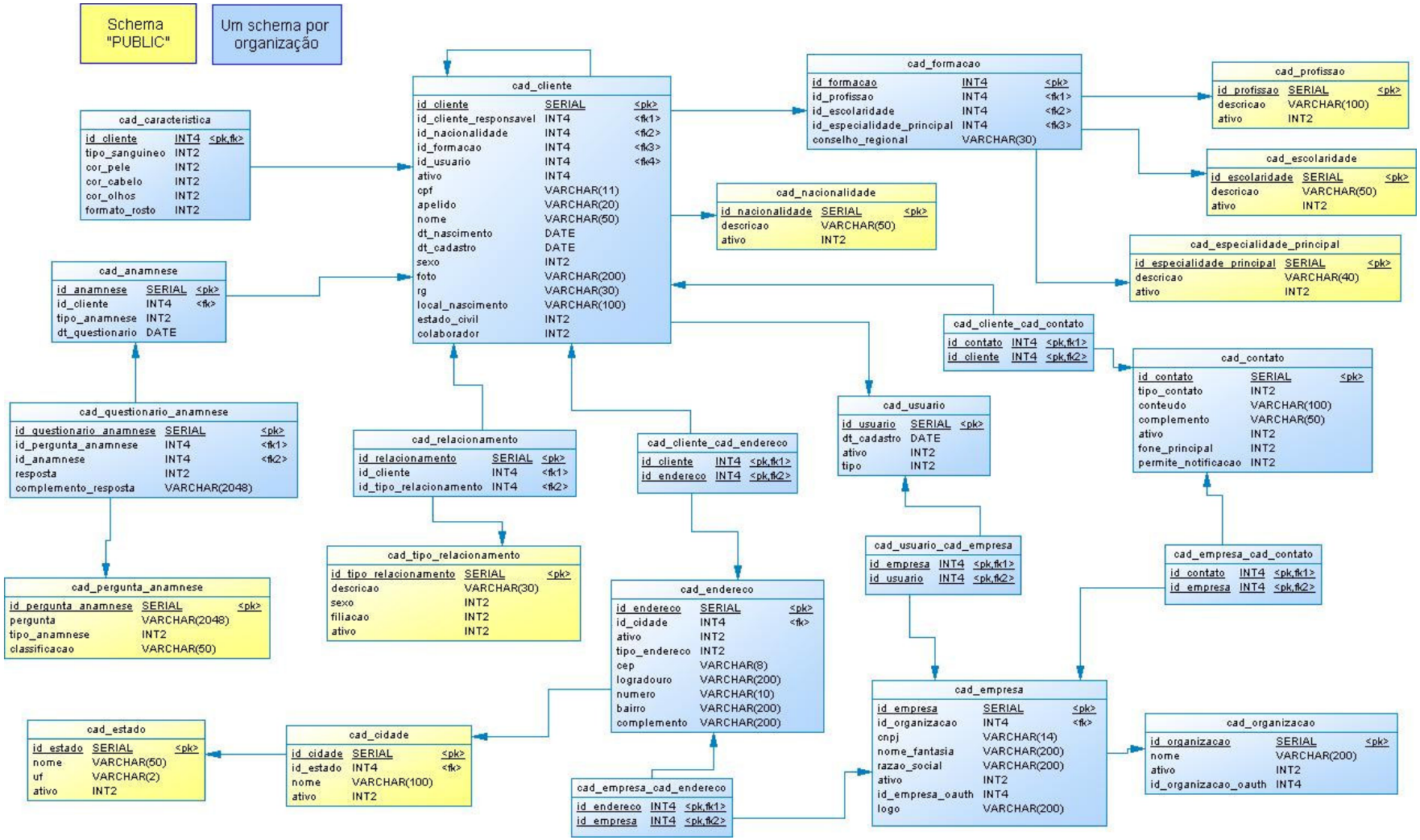

Figura 2 – Diagrama de entidades e relacionamentos do banco de dados do sistema

4.3 APRESENTAÇÃO DO SISTEMA

No primeiro acesso ao sistema, o usuário será direcionado à página de *login* da aplicação (Figura 3). A partir dessa página, poderá acessar o sistema, informando seu e-mail e senha ou poderá acessar as opções de recuperação de senha ou para cadastrar-se.

Acessando a página de recuperação de senha, o usuário encontrará um campo para informar seu *e-mail* e o botão "Esqueci minha senha". Clicando nesse botão, o usuário receberá um *e-mail* contendo uma nova senha.

Acessando a página de cadastro, encontrará apenas dois campos para informar seu nome completo e seu *e-mail*. Esses campos correspondem a um pré-cadastro. Assim que o administrador do sistema liberar seu cadastro para acesso, um *e-mail* será enviado a esse usuário.

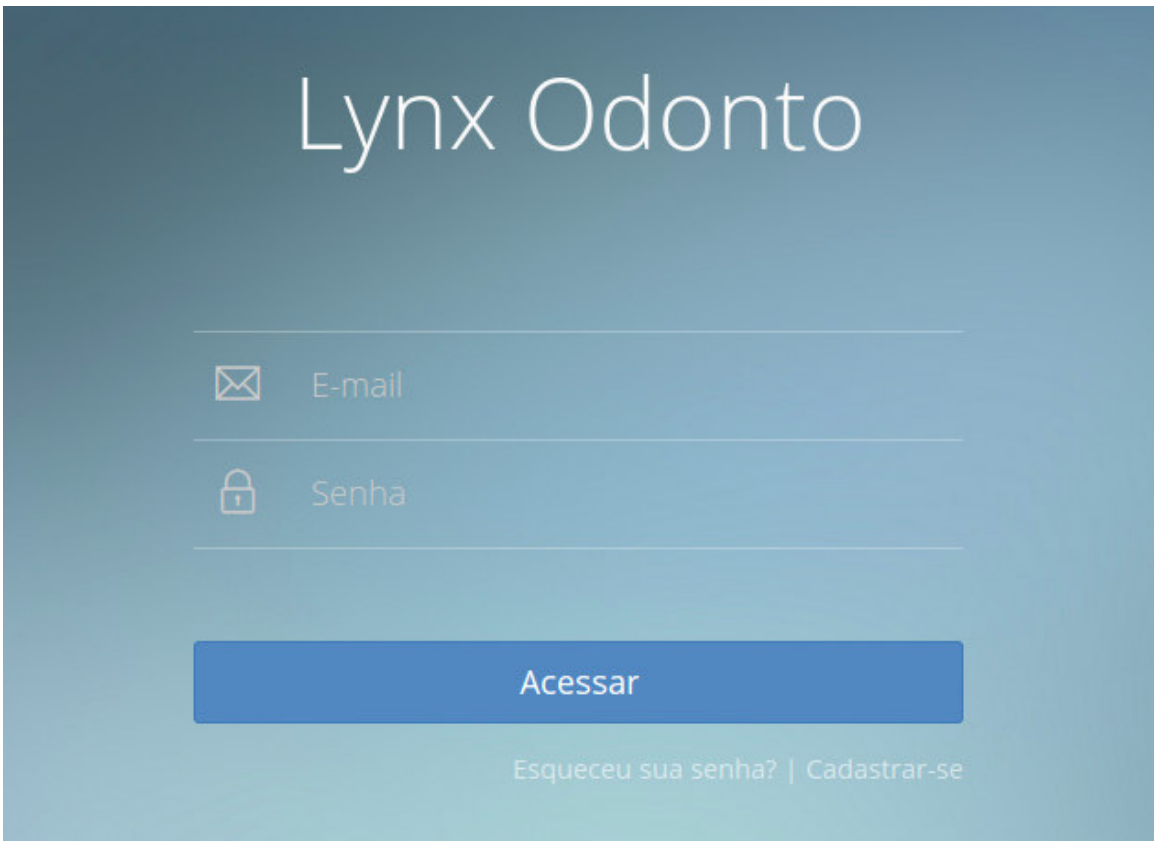

Figura 3 – Página de acesso

Acessando o sistema, o usuário encontrará um leiaute composto de três partes. Uma barra compõe a parte superior do sistema, que contém o nome da empresa, dados de personalização de aparência da aplicação e, à direita, a imagem do usuário logado, que contém opções de acesso ao perfil e página de tarefas do usuário. Do lado esquerdo, o menu,

contendo todas as opções de acesso do sistema, compõe a parte lateral do sistema. Essa parte em específico pode variar para cada usuário logado, pois é montada de acordo com as permissões de acesso de cada usuário. E, por fim, a parte central na qual é carregado o conteúdo de todas as páginas acessadas pelo usuário.

Para facilitar a visualização destas partes na Figura 4, as opções de aparência da página foram alteradas para que cada parte fique com uma cor predominante. A parte superior é representada pela cor azul, a parte lateral em cinza escuro e a parte central, local em que os dados da página estão carregados, está com a cor predominante branca.

Figura 4 – Página inicial

Como está apresentado nas Figuras 5 e 6, as opções de personalização da página são bem diversificadas. É possível minimizar a barra lateral, alterar o formato de exibição da página para ser visualizado em monitores que não possuem tela *widescreen*, alterar o efeito de transição entre as páginas, alterar a orientação de exibição do menu lateral, alterar as cores do leiaute, entre outros.

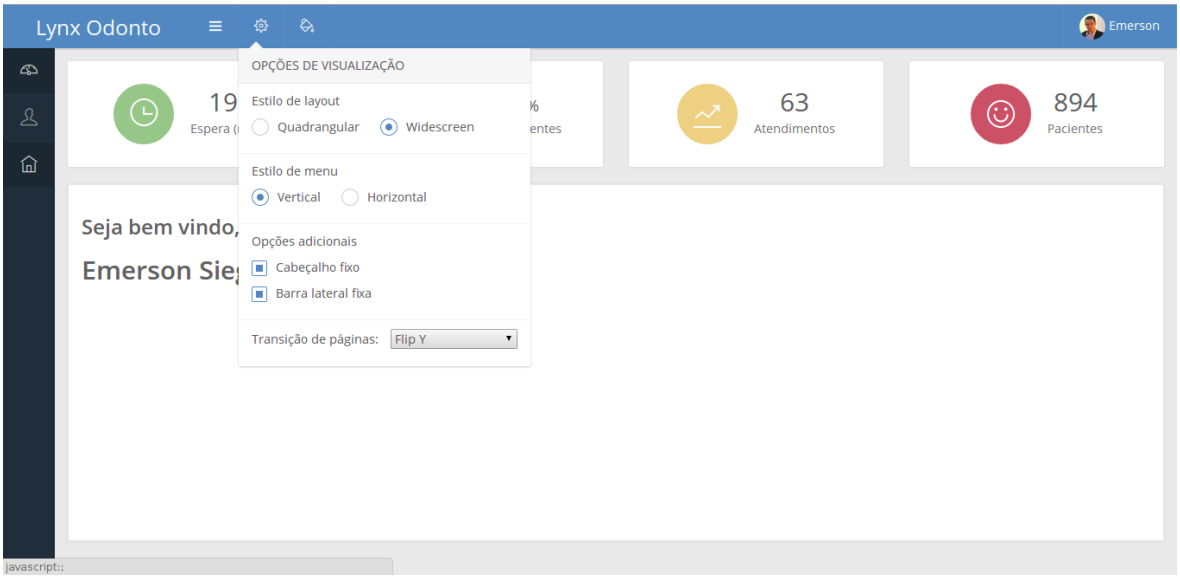

Figura 5 – Opções de visualização

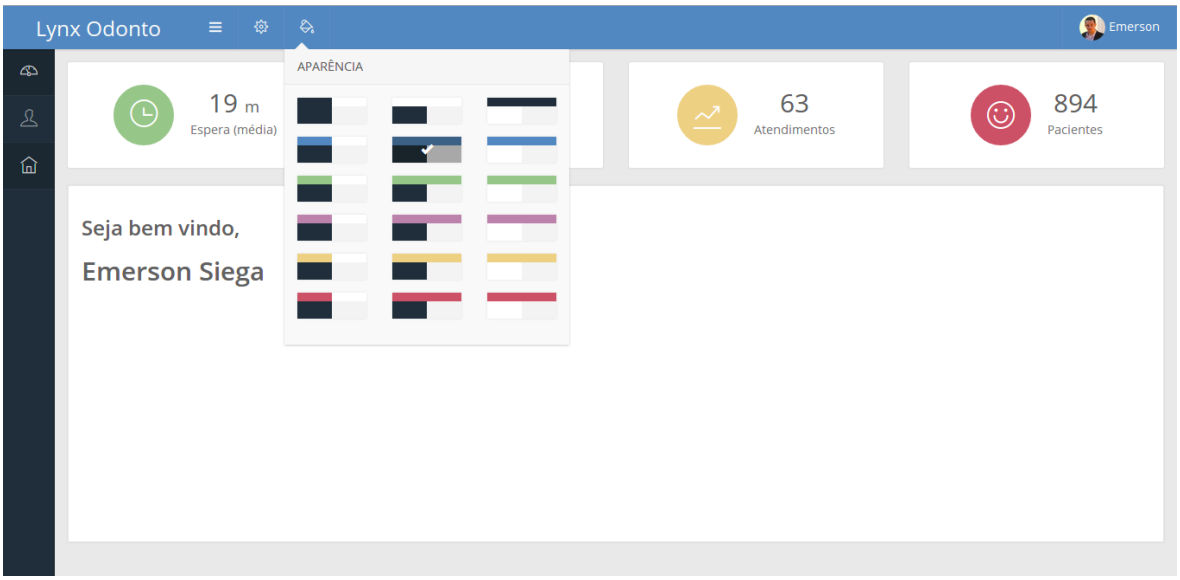

Figura 6 – Cores do leiaute

Na próxima imagem da Figura 7 é apresentada a lista de pacientes da clínica. Essa é uma lista com paginação do lado do servidor para minimizar o tráfego de dados desnecessário, com opções de visualização de 10, 25, 50 e 100 pacientes (linhas da tabela) e navegação entre as páginas. Também possui opções de filtro pelo nome do paciente e uma opção para adicionar novos usuários. Cada paciente da lista possui um botão para edição dos seus dados.

O acesso à página de listagem de pacientes e ao botão para incluir novos pacientes depende de permissão de acesso.

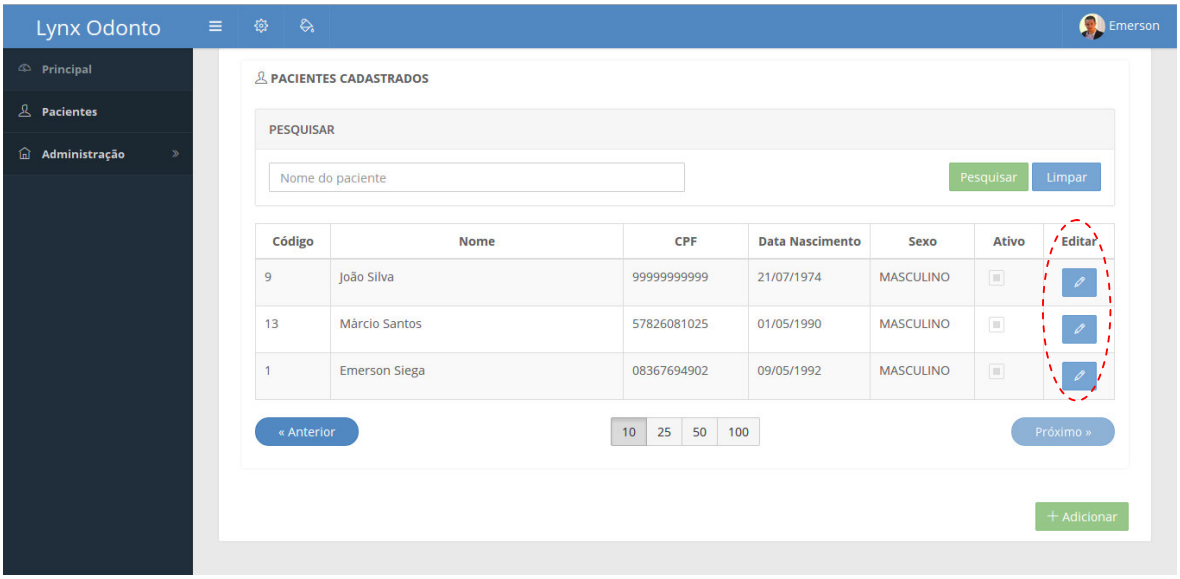

Figura 7 – Listagem de pacientes

Clicando na opção de edição do paciente, área circulada da Figura 7, a página da Figura 8 é apresentada. Nela é exibida uma barra superior, contendo a foto do paciente e as opções de dados principais, contato e endereço. Cada opção altera o conteúdo da página, na qual podem ser editados, respectivamente, os dados básicos do paciente, dados de contato como *e-mail* e telefones e os dados de endereço deste paciente, podendo ser cadastrado mais de um endereço.

Logo abaixo da barra superior, com a opção “Dados Principais” selecionada, é possível editar dados básicos do usuário, bem como sua foto. Para o caso da foto, ela pode ser capturada no momento do cadastro, como mostra a Figura 8, ou pode ser selecionada uma imagem previamente armazenada no computador.

Além dos dados apresentados na imagem, existem outros campos que podem ser preenchidos, como data de nascimento, estado civil e opção para ativar/inativar o paciente.

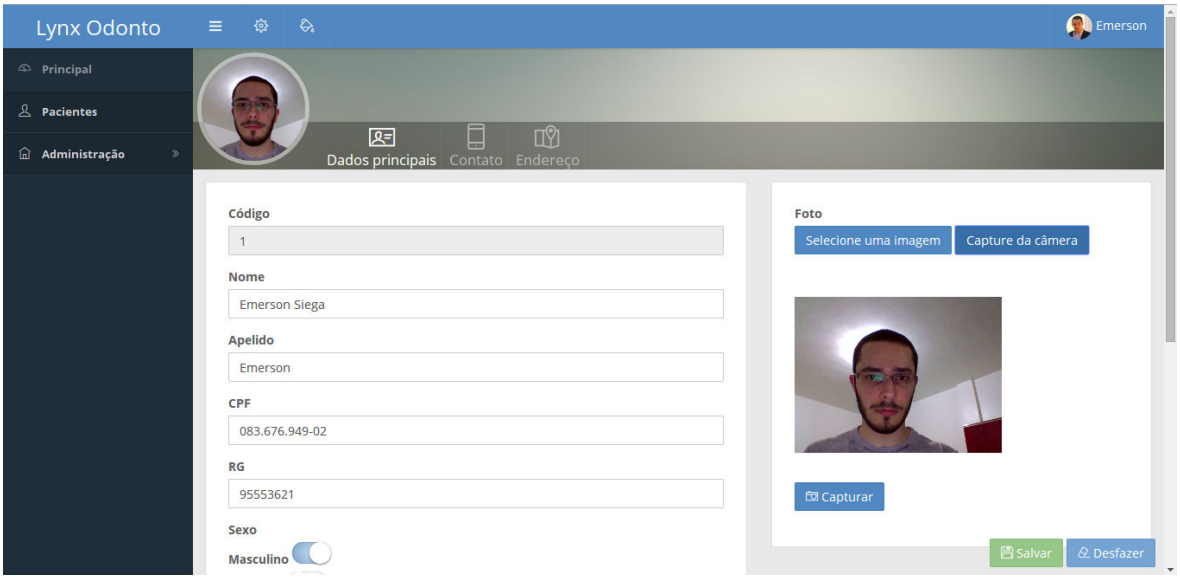

Figura 8 – Edição dos dados principais do paciente

Ainda no cadastro do paciente, estando com a opção de "Contato" selecionada (Figura 9), é apresentada uma lista com os dados de contatos do paciente. É possível adicionar várias informações, como *e-mail*, celular, telefone residencial, telefone comercial e telefone alternativo. Também é possível marcar algumas opções adicionais, como "Fone principal" e "Permite notificação", pois futuramente será realizado o envio de mensagens de texto para o celular dos pacientes, com lembretes de suas consultas.

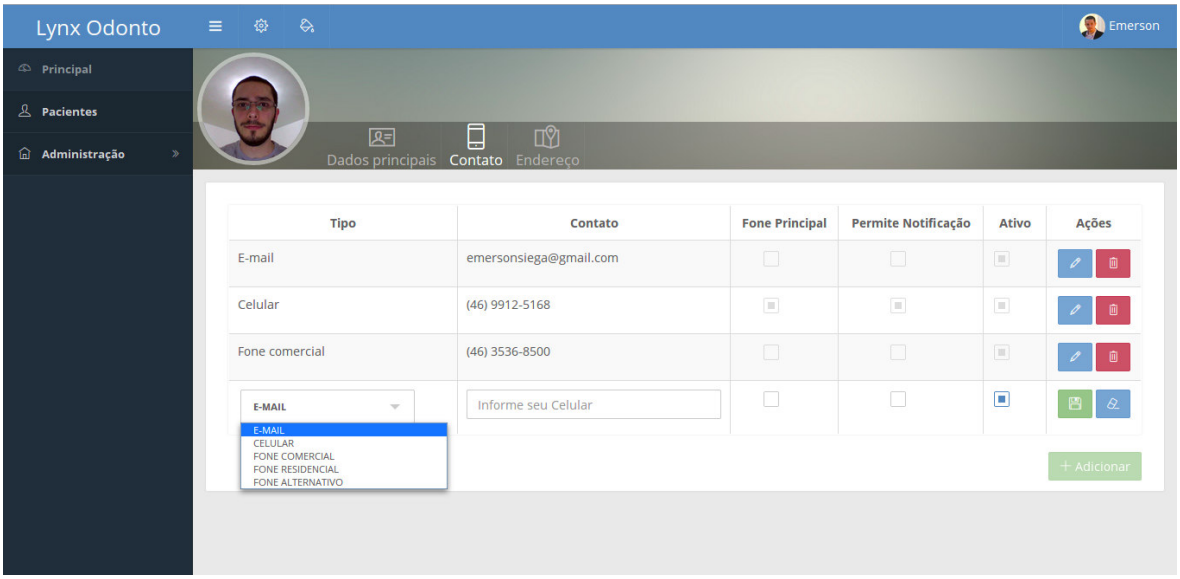

Figura 9 – Edição das informações de contato do paciente

Na Figura 10, ainda dentro do cadastro de pacientes, é exibida a página de edição dos dados de endereços do paciente. Nessa página, como nos dados de contatos, é possível inserir vários registros de endereço. Os tipos de endereço disponíveis são: comercial, residencial, correspondência, cobrança e outro.

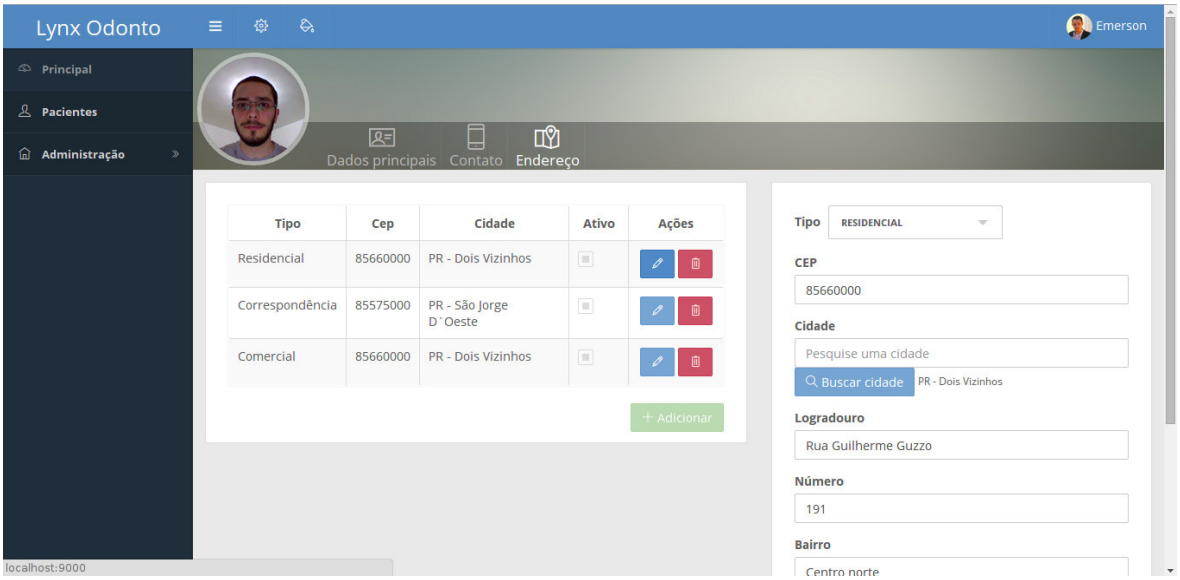

Figura 10 – Edição das informações de endereço do paciente

Na Figura 11 é apresentada a listagem de organizações. Uma organização representa um grupo de clínicas. Normalmente, essa página conterá apenas uma organização e será acessada somente por administradores da aplicação.

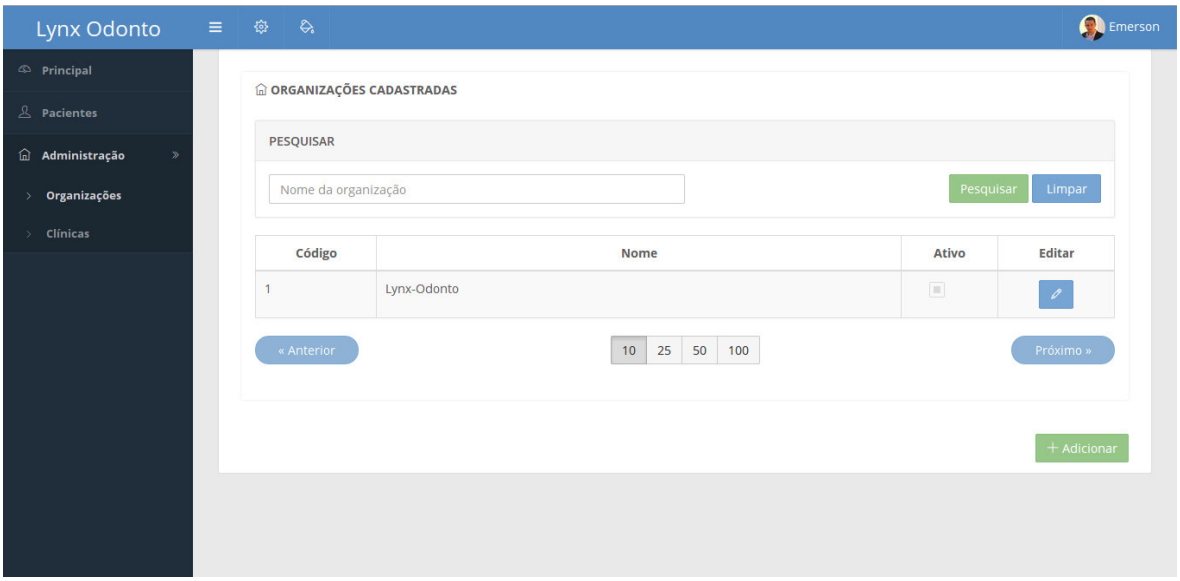

Figura 11 – Listagem de organizações

No contexto do banco de dados, os dados de cada organização estarão alocados em *schemas* diferentes, de maneira que um grupo de clínicas não acesse os dados de outra organização/clínicas. Essa página só poderá ser acessada por usuários que tiverem permissão de acesso.

Na imagem da Figura 12 pode ser visualizada a listagem de clínicas. O usuário logado só poderá acessar essa página se tiver permissão de acesso e só conseguirá acessar os dados da organização à qual está relacionado. Assim como a listagem de pacientes e de organizações. Essa lista é paginada no lado do servidor e pode ser navegada pelas opções de navegação logo abaixo da tabela. Cada linha possui no seu final um botão de edição, para acessar o cadastro da clínica.

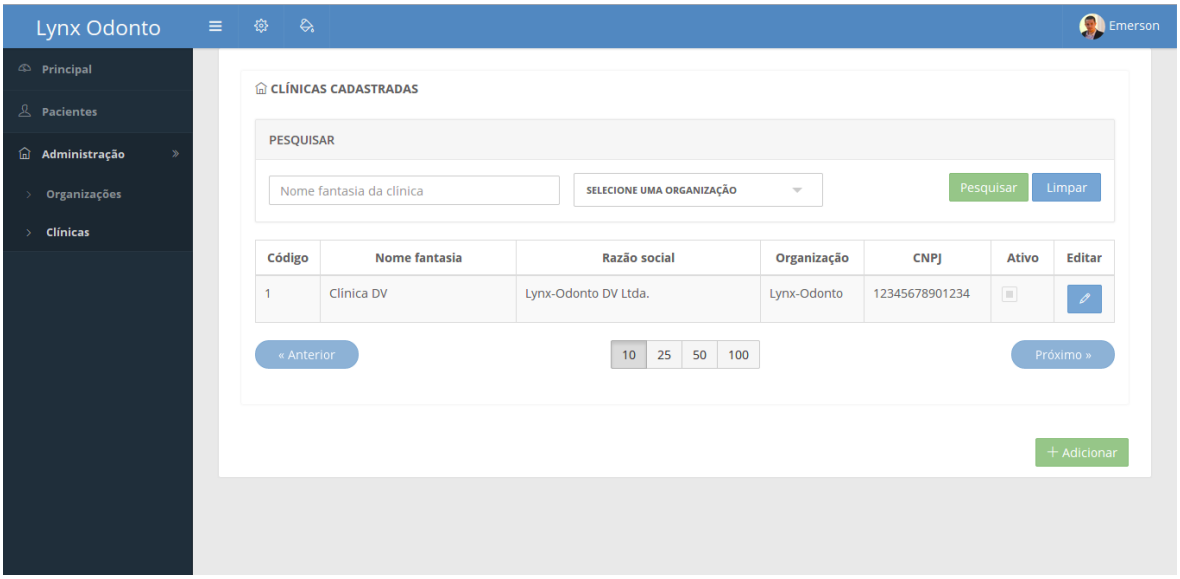

Figura 12 – Listagem de clínicas

Acessando a opção de edição da clínica, mostrada na Figura 12, é possível visualizar um leiaute semelhante ao do cadastro de pacientes, com a logomarca da clínica e as opções para acesso aos dados principais, contato e endereço. Além dessas opções, também existe uma opção adicional para acesso ao cadastro de usuários do sistema, relacionados à clínica. Nesse contexto, cada usuário é um colaborador da clínica e, geralmente, será um dentista, recepcionista, recursos humanos, financeiro, entre outros. Para visualizar e acessar o cadastro de usuários, o usuário logado deve ter direito de acesso a essa opção.

Os cadastros de contatos e endereços da clínica são iguais ao cadastro de pacientes. Esses cadastros foram separados em componentes para serem reutilizados quando necessário, mantendo também um padrão de interface.

Na Figura 13 é apresentada a edição dos dados principais da clínica. Nesse formulário é possível informar os dados básicos da clínica, bem como a logomarca, que pode ser carregada de um arquivo de imagem e recortada conforme mostrado na imagem destacada na Figura 13.

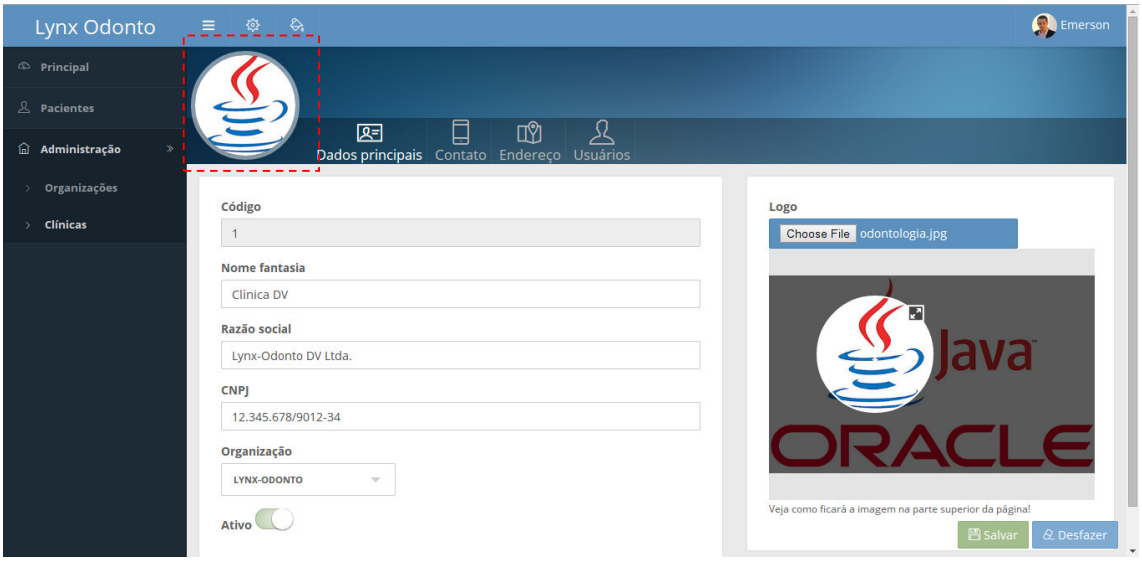

Figura 13 – Edição dos dados principais da clínica

Na Figura 14 é exibida a página na qual é possível gerenciar os usuários relacionados à empresa. Essa página é dividida em duas partes: na primeira parte, à esquerda, estão os usuários já cadastrados e relacionados à empresa. Na segunda parte, à direita, estão os dados do usuário selecionado ou os campos para adição de novos usuários.

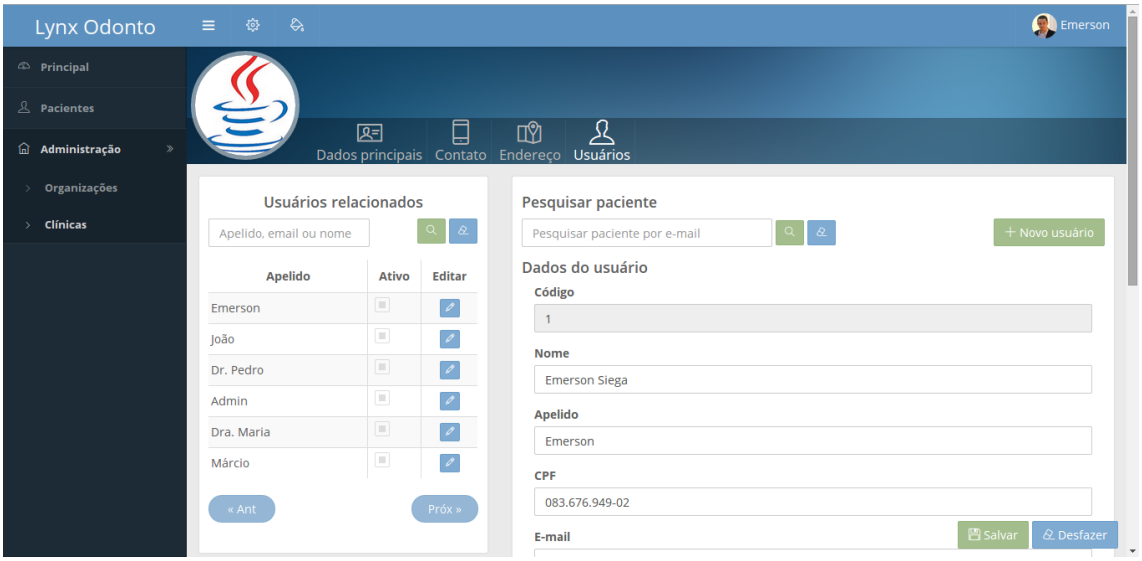

Figura 14 – Página do cadastro de usuários

A lista dos usuários cadastrados também segue o padrão de paginação (Figura 15), porém neste caso são apresentados somente 10 usuários por vez. É possível realizar uma pesquisa em um campo único, no topo da lista, que filtra os usuários por apelido, *e-mail* ou nome. Clicando no botão editar, disponível na linha de cada usuário, os dados do usuário são carregados no painel à direita, no qual é possível editar as informações de acesso do usuário selecionado.

No topo do painel da direita existe um campo de pesquisa, no qual é possível fazer a pesquisa de pacientes previamente cadastrados. Isso é possível, pois os pacientes podem ser "convertidos" em usuários. Essa estratégia foi escolhida, pois os dados básicos de um usuário são praticamente os mesmos dados de um paciente. Desta forma, pode-se utilizar o mesmo cadastro. Caso seja um novo cadastro de usuário (sem paciente previamente cadastrado), pode ser utilizado o botão "Novo usuário".

Logo abaixo dessa pesquisa de pacientes estão os dados do usuário. Nesses dados estão disponíveis campos como: nome, apelido, CPF, *e-mail*, senha e permissões de acesso.

Caso um novo usuário seja criado, será possível acessar seus dados no cadastro de pacientes e preencher informações complementares, como foto, data de nascimento, entre outros.

Figura 15 – Página do cadastro de usuários – Permissões de acesso

A lista de permissões de acesso é gerada dinamicamente. O usuário logado só pode liberar acesso para os mesmos módulos que ele tem acesso. Ou seja, se ele só tem acesso ao cadastro e lista de usuários, só poderá liberar acesso a estes dois módulos para outros usuários.

Nessa página também pode ser definida a senha do usuário. Caso seja de interesse de quem estiver cadastrando, pode ser gerada uma senha automaticamente (hexadecimal de seis dígitos), utilizando o botão no final da página. Ao salvar o usuário, se for um novo cadastro ou se a senha de acesso foi alterada, será enviado um *e-mail* para o usuário, informando-o sobre a mudança/cadastro.

Na Figura 16 pode ser visualizada a página que é apresentada quando o usuário tentar acessar uma página que ele não possui permissão de acesso. Isso acontecerá somente se o usuário tentar acessar uma página diretamente pela URL no *browser,* pois todos os menus estão devidamente protegidos e não são apresentados a quem não tem permissão.

Figura 16 – Página de erro 403

4.4 IMPLEMENTAÇÃO DO SISTEMA

Conforme apresentado na Seção 4.3, várias páginas do sistema possuem controle de acesso e somente usuários logados com *e-mail* e senha e que tenham permissão de acesso podem acessá-las. Para realizar esse controle, foi necessário desenvolver várias classes no *front-end* e *back-end* da aplicação.

Um dos módulos mais importantes desenvolvidos no *front-end* é apresentado na Listagem 1. Esse módulo fica "observando" as mudanças nas rotas (páginas) da aplicação e faz o controle do acesso de acordo com o usuário logado. Se nenhum usuário estiver logado, redireciona para a página de *login*. Se o usuário logado não possuir direito de acesso à página solicitada, será redirecionado para a página de erro 403 (acesso negado), seguindo o padrão de

códigos *Hypertext Transfer Protocol* (HTTP). Um detalhe interessante desse módulo é que, se ele entender que é necessário fazer *login* para acessar a página solicitada, ele armazena essa página em memória e assim que o usuário efetuar o *login* é redirecionado para essa página.

```
'use strict';

lynxApp.run(['$rootScope', '$location', 'authorization', function($rootScope, $location, authorization) {
	var routeChangeRequiredAfterLogin = false;
	var loginRedirectUrl;

	$rootScope.$on('$routeChangeStart', function(event, next) {
		var authorized;

		if (routeChangeRequiredAfterLogin && next.originalPath !== '/signin') {
			routeChangeRequiredAfterLogin = false;
			$location.path(loginRedirectUrl).replace();
		} else if (next.access !== undefined) {
			authorized = authorization.authorize(
				next.access.loginRequired,
				next.access.permissions,
				next.access.permissionCheckType);

			if (authorized === authorization.enums.authorized.loginRequired) {
				routeChangeRequiredAfterLogin = true;
				loginRedirectUrl = getRedirectUrl(next);
				$location.path('/signin');
			} else if (authorized === authorization.enums.authorized.notAuthorized) {
				$location.path('/403').replace();
			}
		}
	});

	/**
	 * Gera URL para redirecionamento, considerando os parâmetros.
	 */
	var getRedirectUrl = function(next) {
		if (next.$$route.keys != undefined && next.$$route.keys.length > 0) {
			var url = next.originalPath;

			next.$$route.keys.forEach(
				function(key) {
					url = url.replace(":" + key.name, next.params[key.name]);
				}
			);

			return url;
		} else {
			return next.originalPath;
		}
	};
}]);
```

Listagem 1 – AuthorizationService.run.js

A Listagem de código 1 possui função importante no sistema, porém não funciona sozinha. É necessário também existir um mapeamento de todas as rotas que serão protegidas, informando quais as permissões mínimas de acesso que um usuário deve ter para acessá-las.

A Listagem de código 2 apresenta uma parte do módulo principal da aplicação, responsável por fazer esse roteamento adequadamente.

```
/**
 * CÓDIGO ACIMA OMITIDO
 */
return $routeProvider
                .when('/', {
                        redirectTo: "/dashboard"
                }).when('/dashboard', {
                        templateUrl: "views/dashboard.html",
                        access: {
                                loginRequired: true
                        }
                }).when('/pacientes', {
                        templateUrl: 'views/pages/paciente/lista.html',
                        controller: 'pacienteController',
                        access: {
                                loginRequired: true,
                                permissions: [permissions.type.LISTA_PACIENTE]
                        }
                }).when('/pacientes/paciente', {
                        templateUrl: 'views/pages/paciente/cadastro.html',
                        controller: 'pacienteController',
                        access: {
                                loginRequired: true,
                                permissions: [permissions.type.CADASTRO_PACIENTE]
                        }
                }).when('/pacientes/paciente/:id', {
                        templateUrl: 'views/pages/paciente/cadastro.html',
                        controller: 'pacienteController',
                        access: {
                                loginRequired: true,
                                permissions: [permissions.type.CADASTRO_PACIENTE,
                                        permissions.type.LISTA_PACIENTE]
                        }
                }).when('/403', {
                        templateUrl: 'views/pages/403.html'
                }).when('/404', {
                        templateUrl: 'views/pages/404.html'
                }).otherwise({
                        redirectTo: '/404'
                });
/**
 * CÓDIGO ABAIXO OMITIDO
 */
```

Listagem 2 – Parte do roteamento feito no app.js

Com essas duas implementações listadas já é possível proteger as páginas e liberar seu acesso somente para usuários autorizados. Porém, há casos em que os elementos da página, como menus, botões e *inputs*, devem ser ocultados para usuários sem privilégios. Para essa finalidade foi desenvolvida uma diretiva chamada "*access*". Essa diretiva valida os direitos de acesso do usuário logado e, de acordo com os parâmetros informados na declaração do

elemento na página HTML, irá mostrar ou esconder as informações da página. As Listagens 3 e 4 mostram a utilização dessa diretiva e sua implementação, respectivamente.

```
<div class="nav-wrapper">
  <ul id="nav" class="nav" data-ng-controller="NavCtrl">
    <li>
      <a href="#/dashboard">
            <i class="ti-dashboard"></i>
            <span>Principal</span>
      </a>
    </li>
    <li access="LISTA_PACIENTE,CADASTRO_PACIENTE"
      access-dont-remove="true">
      <a href="#/pacientes">
            <i class="ti-user"></i>
            <span>Pacientes</span>
      </a>
    </li>
    <li access="LISTA_ORGANIZACAO,CADASTRO_ORGANIZACAO,LISTA_CLINICA,
      CADASTRO_CLINICA,CADASTRO_USUARIO,LISTA_USUARIO"
      access-dont-remove="true">
      <a href="">
            <i class="ti-home"></i>
            <span>Administração</span>
      </a>

      <ul>
            <li access="LISTA_ORGANIZACAO,CADASTRO_ORGANIZACAO"
              access-dont-remove="true">
                  <a href="#/organizacoes">
                        <i class="ti-angle-right"></i>
                        <span>Organizações</span>
                  </a>
            </li>
            <li access="LISTA_CLINICA,CADASTRO_CLINICA,
              CADASTRO_USUARIO,LISTA_USUARIO"
              access-dont-remove="true">
                  <a href="#/clinicas">
                        <i class="ti-angle-right"></i>
                        <span>Clínicas</span>
                  </a>
            </li>
        </ul>
    </li>
  </ul>
</div>
```

Listagem 3 – Controle de acesso utilizando diretiva "*access*"

```
lynxApp.directive('access', ['authorization', function(authorization) {
    return {
        restrict: 'A',
        link: function(scope, element, attrs) {
            var makeVisible = function() {
                element.removeClass('hidden');
            },
            makeHidden = function(removeElement) {
                element.addClass('hidden');
                if (attrs.accessDontRemove == undefined || attrs.accessDontRemove == "false") {
                    element.remove();
                }
            },
            determineVisibility = function(resetFirst) {
                if (resetFirst) {
                    makeVisible();
                }
                var result = authorization.authorize(true, roles, attrs.accessPermissionType);
                if (result === authorization.enums.authorized.authorized) {
                    makeVisible();
                } else {
                    makeHidden();
                }
            },
            roles = attrs.access.split(',');

            if (roles[0].length == 0) {
                roles = undefined;
            }

            // Evento disparado quando é concluído o login e já existem dados do usuário logado
            scope.$root.$on("login-done", function() {
                determineVisibility(true);
            });
        }
    };
}]);
```

Listagem 4 – Diretiva "*access*"

Em conjunto com todos esses módulos desenvolvidos no *front-end* foi necessário desenvolver um projeto paralelo, que segue os padrões especificados no *framework* OAuth 2.0. Esse projeto, que é somente *backend*, implementa um sistema de autenticação genérico que possibilita fazer o controle de aplicações, usuários, relacionamento entre usuários, empresas e organizações, direitos de acesso de usuários e dos *tokens* de acesso de usuários (chaves de acesso às aplicações cadastradas). O diagrama de entidades e relacionamentos criado para essa aplicação está disponível na Figura 1 da Seção 4.2.

Na Listagem 5 é possível verificar um trecho de código desse sistema de autenticação. No trecho apresentado, o sistema de autenticação valida as informações que são enviadas durante o *login* do usuário na aplicação. Entre algumas das validações realizadas estão as seguintes:

- Validação do código de aplicação enviado na requisição (se pertence a uma aplicação cadastrada no banco de dados);
- Validação do usuário enviado na requisição (se existe no banco de dados);
- Validação de senha do usuário (senha enviada corresponde à senha cadastrada);
- Validação de permissão de acesso (vínculo) do usuário à aplicação informada.

Caso a tentativa de *login* não seja inválida para nenhuma dessas validações, então será gerado um *token* de acesso relacionado ao usuário e à aplicação e será persistido no banco de dados. No retorno desse serviço são enviados o próprio *token* de acesso e o tempo de validade do referido *token*.

```
public abstract class AbstractAuthorizationController implements AuthorizationController {
   /**
    * CÓDIGO ACIMA OMITIDO
    */
   @Override
   public Response authorize( OAuthTokenRequest oauthTokenRequest )
         throws OAuthSystemException {

      Aplicacao aplicacao = findAplicacao( oauthTokenRequest.getClientId() );
      AplicacaoResponse aplicacaoResponse = checkAplicacao( aplicacao,
                  oauthTokenRequest.getClientSecret() );

      if ( !aplicacaoResponse.equals( AplicacaoResponse.SUCESSO ) ) {
         return ResponseBuilder.buildInvalidClient( aplicacaoResponse );
      }

      Usuario usuario = findUsuario( oauthTokenRequest.getUsername() );
      UsuarioResponse usuarioResponse = checkUsuario( usuario, oauthTokenRequest.getPassword() );

      if ( !usuarioResponse.equals( UsuarioResponse.SUCESSO ) ) {
         if ( oauthTokenRequest.getParam( OAuth.OAUTH_GRANT_TYPE )
                  .equals( GrantType.PASSWORD.toString() ) ) {
            return ResponseBuilder.buildInvalidUserPassResponse();
         } else if ( oauthTokenRequest.getParam( OAuth.OAUTH_GRANT_TYPE )
                  .equals( GrantType.REFRESH_TOKEN.toString() ) ) {
            return ResponseBuilder.buildInvalidUserPassResponse();
         }
      }

      UsuarioAplicacaoResponse userAppResponse = checkUsuarioAplicacao( usuario, aplicacao );

      if ( userAppResponse.
                  equals( UsuarioAplicacaoResponse.USUARIO_SEM_ACESSO_APLICACAO ) ) {
         return ResponseBuilder.buildInvalidPermissaoAplicacaoResponse();
      }
      Token token = tokenController.addToken( aplicacao, usuario, Integer.parseInt( expiresIn ) );

      return ResponseBuilder.acceptRequest( token.getTokenAcesso(), expiresIn );
   }
   /**
    * CÓDIGO ABAIXO OMITIDO
    */
}
```

Listagem 5 – Rotina de autenticação do sistema OAuth

Na Seção 4.2 foram apresentadas algumas páginas que contém tabelas com paginação no lado do servidor. Para efetuar essa paginação, as páginas devem enviar ao servidor dados como página atual e quantos itens devem ser listados (por padrão, as páginas do sistema possuem limites de 10, 25, 50 e 100 itens). Para garantir que nunca serão realizadas chamadas de serviços que possuem paginação com valores muito elevados ou sem informar o limite de itens a serem pesquisados, foi criada uma classe seguindo o *Design Pattern* chamado *Interceptor*. Na Listagem de código 6 é apresentada essa classe, que, quando utilizada corretamente, valida os parâmetros enviados para métodos de consulta e, caso necessário, realiza o ajuste do limite de itens.

```
@Interceptor
public class PageValidatorInterceptor  implements Serializable{
	@AroundInvoke
	public Object intercept(InvocationContext  context) throws Exception {
		Object[] parameters  = context.getParameters();
		if (parameters  != null) {
			for (int i = 0; i < parameters.length;  i++) {
				Object object  = parameters[i];

				if (object instanceof Page) {
					Page page = (Page) object;

					if (page.getLimit()  <= 0) {
						page.setLimit(50);
					} else if (page.getLimit()  > 100) {
					   page.setLimit( 100 );
					}
				}
			}
		}

		Object object = context.proceed();
		return object;
	}
}
```

Listagem 6 – Interceptor de paginação

Na Listagem 7 é apresentado um exemplo de utilização do *Interceptor* *PageValidatorInterceptor*, exibido na Listagem 6, em uma pesquisa de pacientes.

```
@Stateless
public class ClienteEJBImpl implements ClienteEJB {
   @Inject
   private ClienteDAO clienteDAO;
   @Inject
   private ClienteConverter clienteConverter;

   @Override
   @Interceptors( { PageValidatorInterceptor.class } )
   public List<ClienteDTO> findAll( Page page ) {
      List<Cliente> clientes = clienteDAO.findAll( page );

      List<ClienteDTO> response = clienteConverter.criarDTOList( clientes );
      return response;
   }

   /**
    * CÓDIGO ABAIXO OMITIDO
    */
}
```

Listagem 7 – Utilização do interceptor de paginação

Outro recurso muito importante da aplicação que foi citado na Seção 4.2 é a divisão dos dados de organizações em diferentes *schemas* no banco de dados. Para realizar essa divisão, o sistema acessará o servidor de autenticação para, a partir do *token* de acesso do usuário, buscar seu perfil (*schema*). A partir disso, o nome do *schema* é armazenado em uma classe local para futura utilização. Para utilizar esse recurso de divisão em *schemas*, foi necessário configurar três propriedades extras de conexão com o banco de dados, apresentadas na Listagem 8.

```
<!-- OUTRAS CONFIGURACOES -->
<properties>
        <property
                name="hibernate.multi_tenant_connection_provider"
                value="br.com.apus.lynx.dao.impl.MultiTenantConnectionProviderImpl" />

        <property
                name="hibernate.tenant_identifier_resolver"
                value="br.com.apus.lynx.dao.impl.CurrentTenantIdentifierResolverImpl" />

        <property name="hibernate.multiTenancy" value="SCHEMA" />
</properties>
```

Listagem 8 – Trecho das configurações do Persistence.xml

A propriedade "*hibernate.multiTenancy*" indica que será utilizada a estratégia de divisão em *schemas*. Especificando essa propriedade, é obrigatório especificar uma classe na propriedade "*hibernate.multi_tenant_connection_provider*", a qual irá prover as conexões com o banco de dados, já aplicando o *schema* utilizado na conexão. E por fim, foi necessário definir a classe da propriedade "*hibernate.tenant_identifier_resolver*", que é a classe

responsável por informar ao *connection provider* qual o *schema* utilizado. Nas Listagens 9 e 10 estão trechos das duas classes utilizadas nessas configurações.

```
public class MultiTenantConnectionProviderImpl
        implements MultiTenantConnectionProvider, Stoppable {

    @Override
    public Connection getConnection( String tenantIdentifier ) throws SQLException {
        final Connection connection = getAnyConnection();

        try {
            connection.createStatement().execute( "SET SCHEMA '" + tenantIdentifier + "'" );
        } catch ( SQLException e ) {
            throw new HibernateException(
                    "Não foi possível alterar a conexão JDBC para o schema ["
                    + tenantIdentifier + "]", e );
        }

        return connection;
    }
}
```

Listagem 9 – Trecho de código da classe MultiTenantConnectionProviderImpl

O trecho de código apresentado na Listagem 10 utiliza uma classe específica que contém a informação do *schema* do usuário.

```
public class CurrentTenantIdentifierResolverImpl implements CurrentTenantIdentifierResolver {

    @Override
    public String resolveCurrentTenantIdentifier() {
        String tenant = ThreadLocalUtil.getThreadVariable( "schema" ).toString();

        return tenant;
    }

}
```

Listagem 10 – Trecho de código da classe CurrentTenantIdentifierResolverImpl

Desta forma, todas as operações realizadas por usuários de clínicas diferentes ficarão totalmente isoladas umas das outras e as clínicas só terão acesso às informações que lhe forem destinadas.

5 CONCLUSÃO

O principal objetivo da solução gerada pelo presente trabalho é fornecer uma base para o que se tornará um sistema de grande porte e complexo para o gerenciamento de clínicas odontológicas, que será disponibilizado em ambiente *cloud*.

Com o desenvolvimento desta solução, foi possível obter um grande ganho de conhecimento e experiência no desenvolvimento de sistemas utilizando tecnologias como Java, HTML5 e AngularJS e na disponibilização de aplicações como serviço. Também foi obtido um bom conhecimento no que diz respeito ao fluxo de atendimento de uma clínica odontológica, seu processo de atendimento e agendamento de consultas com pacientes e sua organização em um consultório dividido por vários dentistas.

Durante o processo de desenvolvimento, várias complicações e dificuldades foram superadas. Dentre elas é importante destacar a complexidade gerada pela utilização de uma arquitetura na qual todo o controle de fluxos e páginas do *front-end* é feita manualmente, apenas utilizando-se de JavaScript. Para um desenvolvedor *back-end* isso se torna algo bem complexo. Outro ponto que deve ser destacado é a não utilização de *frameworks* prontos para controle de acesso da aplicação. Essa foi uma parte bem demorada do desenvolvimento, que gerou vários pequenos problemas que foram custosos e complexos de serem resolvidos.

Entretanto, apesar das dificuldades encontradas, os resultados gerados foram satisfatórios. Toda a parte de segurança da aplicação foi atendida com o desenvolvimento de um sistema a parte para fazer o controle de autenticação e autorização de usuários. O banco de dados ficou bem estruturado com a separação por *schemas*. O *front-end* da aplicação ficou extremamente maleável e desvinculado dos serviços, podendo ser facilmente personalizado.

A solução gerada será utilizada conforme o planejamento inicial, na qual serão acrescentados novos módulos e recursos que, de forma gradativa, se tornarão um sistema completo e serão utilizados para proporcionar o gerenciamento total de uma rede de consultórios odontológicos.

REFERÊNCIAS

AMANATULLAH, Yanuarizki; LIM, Charles; IPUNG, Heru Purnomo; JULIANDRI, Arkav. **Toward cloud computing reference architecture: cloud service management perspective.** In: International Conference on ICT for Smart Society (ICISS), 2013. p. 1-4.

BEHL, Akhil; BEHL, Kanika. **An analisys of cloud computing sucurity issues**. World Congress on Information and Communication Technologies (WICT), 2012, p. 109-114.

CLOUD SECURITY ALLIANCE. Disponível em: <http://www.cloudsecurityalliance.org>. Acesso em: 19 mar. 2015.

DIKAIAKOS, Marios D.; PALLIS, George; KATSAROS, Dimitrios; MEHRA, Pankaj; VAKALI, Athena. **Cloud computing: distributed internet computing for it and scientific research**, IEEE Internet Computing, Published by the IEEE Computer Society, September/October 2009, p. 10-13.

HARAUZ, John; KAUFMAN. Lorti M.; POTTER, Bruce. **Data security in the world of cloud computing**. IEEE Security & Privacy, Copublished by the IEEE Computer and Reliability Societies, July/August 2009, v. 7, n.4, p. 61-64.

JADEJA, Yashpalsinh; MODI, Kirit. **Cloud computing - concepts, architecture and challenges**. International Conference on Computing, Electronics and Electrical Technologies (ICCEET), 2012, p. 887-880.

KRUCHTEN, Philippe. The 4+1 view model of architecture, **IEEE Software**, v. 12, n. 6, 1995, p. 45-50.

LIU, Wentao. **Research on cloud computing security problem and strategy**, 2012, p. 1216-1219.

NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY. **Computer security resource center**. Disponível em: < www.csrc.nist.gov>. Acesso em: 20 mar. 2015.

SHAIKH, Farhan Bashir; HAIDER, Sajjad. **Security threats in cloud computing**. 6th International Conference on Internet Technology and Secured Transactions, 2011, p. 214-219.

SHETTY, Sachin. **Auditing and analysis of network traffic in cloud environment**. IEEE Computer Society, 2013, p. 260-267.

YAMINI, R. **Power management in cloud computing using green algorithm**. IEEE-International Conference on Advances in Engineering, Science and Management (ICAESM - 2012), 2012, p. 128-133.