UNIVERSIDADE FEDERAL DE PERNAMBUCO
CENTRO DE INFORMÁTICA

UMA SOLUÇÃO PARA UM SERVIÇO MULTI-TENANCY DE AUTENTICAÇÃO, AUTORIZAÇÃO E CONTROLE DE ACESSO DE USUÁRIOS

JOSÉ BARBOSA DA SILVA NETO

Trabalho de Graduação

Recife
2018

UNIVERSIDADE FEDERAL DE PERNAMBUCO
CENTRO DE INFORMÁTICA

JOSÉ BARBOSA DA SILVA NETO

UMA SOLUÇÃO PARA UM SERVIÇO MULTI-TENANCY DE AUTENTICAÇÃO, AUTORIZAÇÃO E CONTROLE DE ACESSO DE USUÁRIOS.

Trabalho apresentado ao Centro de Informática da Universidade Federal de Pernambuco como requisito parcial para obtenção do grau de Bacharel em Sistemas de Informação.

Professor Orientador: Vinicius Cardoso Garcia

Recife
2018

RESUMO

A arquitetura Multi-Tenancy ou multi-inquilino, em português, é uma das arquiteturas utilizada na construção de um SaaS. Este modelo proporciona com que uma mesma aplicação possa ser utilizada por clientes diferentes compartilhando a mesma base de código e a mesma infraestrutura no qual a aplicação está inserida [1].

De acordo com o modelo arquitetural utilizado, diferentes estratégias e soluções poderão ser aplicadas para garantia do isolamento e segurança dos dados transitados pela aplicação.

A partir do modelo arquitetural *multi-tenant* adotado este trabalho se propõe a realizar um estudo sobre a arquitetura para propor soluções e estratégias que visem o isolamento dos dados bem como sua segurança neste modelo arquitetural para a construção de uma solução para um serviço de autenticação, autorização e controle de acesso de usuários.

Palavras-chave: Software como Serviço, Segurança da Informação, Controle de Acesso, Arquitetura de Software.

ABSTRACT

The Multi-tenancy or multi-tenant architecture, in Portuguese, is one of the architectures used in the construction of a SaaS. This model provides that the same application can be used by different clients sharing the same code base and the same infrastructure in which the application is inserted [1].

According to the architectural model used, different strategies and solutions can be applied to guarantee the isolation and security of the data carried by the application.

Based on the adopted multi-tenant architectural model, this work proposes to carry out a study on the architecture to propose solutions and strategies that aim at the isolation of the data as well as its security in this architectural model for the construction of a solution for an authentication service, authorization and access control of users.

Keywords: Software as a Service, Information Security, Access Control, Software Architecture.

AGRADECIMENTOS

Gostaria de agradecer a todos que diretamente ou indiretamente contribuíram não apenas para a elaboração deste trabalho como também na minha vida pessoal.

Quero agradecer em especial aos meus pais Manoel Barbosa e Graciete Garcez que sempre me apoiaram e incentivaram na minha vida acadêmica e profissional.

Agradeço também ao meu orientador Vinicius Cardoso, pelo suporte e apoio durante todo o decorrer do curso.

Agradeço também aos meus colegas de classe Vitória Maciel, Valdi Ferreira, Victor Nunes e Pedro Clericuzi que estiveram comigo durante todo o curso, nos momentos mais difíceis aos mais descontraídos.

Por fim agradeço também a todos os meus colegas de turma que contribuíram para minha formação acadêmica.

SUMÁRIO

1. INTRODUÇÃO

O conceito de Software as a Service (SaaS) ou Software como Serviço vem se difundindo e chamando atenção da indústria de software. Grandes empresas como Google, Amazon e Netflix utilizam esse modelo de serviço pelo qual os usuários contratam apenas o serviço.

Esse modelo caracteriza-se pela diminuição de custos e por proporcionar maior escalabilidade na aquisição de novos usuários ou inquilinos. Apesar de todos os serviços serem acessados via *web browser* e utilizarem o mesmo local para gerenciamento e utilização, a arquitetura e estratégia empregadas para processamento e gerenciamento das informações dessas soluções não são necessariamente iguais. Ambas são definidas de acordo com o produto a ser construído bem como as políticas de prestação de serviço oferecida por cada produto [2].

O aumento da competitividade no mercado acelerou o processo de transição para construção de aplicações com modelos mais complexos que suprissem a demanda de escalabilidade e seguranças das empresas, tanto no sentido de aquisições de novos clientes bem como no poder computacional dos serviços oferecidos [3].

Para esse novo modelo, a arquitetura de software denominada Multi-tenancy ou multi-inquilino ganhou destaque no processo de criação de aplicações do tipo SaaS possibilitando uma maior escalabilidade e ganho de mercado.

A partir dos pontos levantados este trabalho se propõe a construir um serviço de autenticação, autorização e controle de acesso de usuários em uma arquitetura *multi-tenancy*.

1.2 MOTIVAÇÃO

Baseado na competitividade do mercado a segurança é um fator crucial para a sobrevivência das empresas, atualmente soluções de Single Sign-On como a do Google e do Facebook que possibilita a autenticação dos usuários utilizando as contas de seus.

Apesar de soluções como a do Google e Facebook possibilitarem o gerenciamento de autenticação elas ainda são soluções simples quando diante de cenários onde aplicações mais complexas necessitam de um gerenciamento de segurança não somente da autenticação mas como da autorização e o acesso de controle.

Aplicações como o *Docker Swarm*[1] que é um gerenciador de clusters de containers utilizando o *Docker*[2] que não possui um módulo ou um serviço de gerenciamento de acesso de usuários necessita de uma solução mais complexa que apenas a autenticação de um determinado usuários.

Com base nos problemas levantados e a necessidade observada, notou-se a necessidade da realização de um estudo que desse fundamentos para a criação de um serviço de autenticação, autorização e gerenciamento de acesso e controles de usuários para que aplicações de terceiros possam gerenciar seus módulos de segurança e acesso a suas aplicações através deste serviço.

[1] https://docs.docker.com/engine/swarm/
[2] http://docker.io/

1.3 OBJETIVOS

Nesta seção serão apresentados os objetivos geral e específicos.

1.3.2 OBJETIVO GERAL

Este trabalho tem como principal objetivo a construção de um serviço de gerenciamento de autenticação, autorização e controle de acesso em uma arquitetura multi-tenancy.

1.3.3 OBJETIVO ESPECÍFICOS

- Realizar um estudo sobre a arquitetura *multi-tenancy*.
- Investigar estratégias para a segurança e isolamento dos dados para a arquitetura multi-tenancy.
- Propor e implementar uma solução para o gerenciamento de autenticação, autorização e acesso de usuários para uma arquitetura multi tenancy.

1.3.4 ESTRUTURA DO TRABALHO

Este trabalho apresenta ao todo 4 capítulos. O capítulo 2 apresenta todos os conceitos necessários para o entendimento da pesquisa realizada de maneira que o capítulo 3 aborda todo o processo de construção da solução proposta e o capítulo 4 os resultados obtidos. Por fim, o capítulo 5 apresenta as considerações finais e trabalhos futuros.

2 REFERENCIAL TEÓRICO

Neste capítulo serão abordadas as definições e conceitos base, de modo que o problema investigado possa ser contextualizado e melhor compreendido, tais como Computação em Nuvem, Software como Serviço e a arquitetura *Multi tenancy*.

2.1 COMPUTAÇÃO EM NUVEM

Define-se *Cloud Computing* como um modelo ubíquo, conveniente para acesso de recursos computacionais configuráveis que podem ser providos e entregues rapidamente com o menor esforço possível sem a necessidade de interação com os provedores de serviços (*National Institute of Standards and Technology*, 2018).

Na computação em nuvem, Software as a Service é um dentre os modelos que podem ser utilizado para construção de serviços baseados em nuvem. Esse modelo de software é vendido através de licença, diferentemente dos modelos de negócios mais antigos, onde a obtenção dava-se por meio da cópia do software.

Esse modelo para provimento de recursos em nuvem trouxe consigo alguns benefícios, como: redução de custos e a escalabilidade. O primeiro baseia-se no fato de que na computação em nuvem o investimento inicial em infraestrutura é mínimo, necessário apenas para garantir a disponibilidade do serviço, e de modo que os clientes possam usufruir de acordo com a sua necessidade. O segundo refere-se a possibilidade no qual novas máquinas poderão ser inicializadas e configuradas automaticamente para que o serviço no qual será disponibilizado não fique fora do ar em casos de aumento de acesso.

Soluções construídas especificamente para serem rodadas na nuvem comumente se utilizam do modelo SaaS, além das características já citadas anteriormente esse modelo também apresenta uma alta velocidade no provisionamento e na entrega de novas atualizações.

De acordo com o livro *Architecture Strategies for Catching the Long Tail* [7] o modelo *SaaS* apresenta 4 níveis de maturidade, esses níveis determinam o grau de

maturidade no qual o software que foi construído está se utilizando das características que devem ser inerentes para que a mesma seja classificada como uma solução para a nuvem, conforme apresentado na ilustração criada pelo escritor *Frederick Chong* e *Gianpaolo Carraro* no qual foram propostos os 4 níveis de maturidade [5].

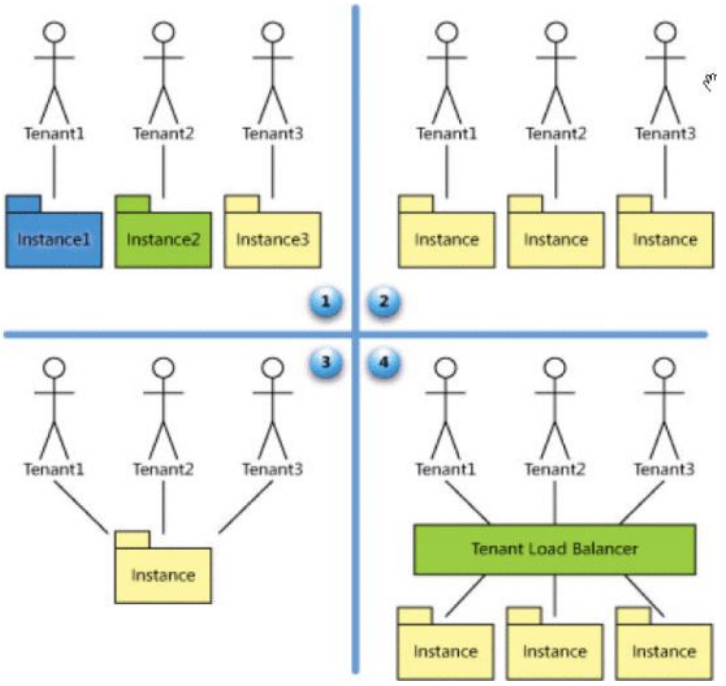


Figura 1 - SaaS níveis de maturidade. [7]

O modelo de maturidade nível 1 chamado de *Ad Hoc* ou *Custom* que seriam sinônimos para descreverem as customizações distintas de cada cliente para a aplicação, nesse nível aplicações idênticas são copiadas e customizadas para que os clientes possam se utilizar, esse modelo é muito comum por ser simples de ser implementado porém tanto sua atualização bem como a distribuição de suas atualizações não ocorrem de forma escalável.

O modelo de maturidade nível 2 chamado de Configurable ou configurável é muito similar ao modelo 1, ambos continuam com uma instância para cada cliente porém no nível 2 todos os clientes se utilizam de uma cópia exata da aplicação porém diferente do nível dois todas as customizações em cada instância é feita pelo próprio cliente através da interface.

O nível 3, *Configurable* e *Multi-Tenant-Efficient*, este modelo apresenta quase todas as características do nível 2 porém uma única instância é utilizada, para esse nível todos os clientes começam a se utilizam de uma única instância com uma única aplicação, desse modo a distribuição e entrega de atualizações para todos os clientes começa a ser escalável tendo em vista que a atualização em uma única instância iria ser replicadas para todos ao mesmo tempo porém nesse nível todos os recursos da instância são compartilhados por todos os clientes dando a possibilidade de instabilidades durante sua utilização.

O nível 4, *Scalable*, *Configurable* e *Multi-Tenant-Efficient*, esse nível é o mais complexo a se chegar, esse nível apresenta todas as características do nível 3 porém ao invés dos clientes acessarem diretamente uma única instância da aplicação nesse nível uma outra camada é adicionada que é o balanceador de carga, esse balanceador tem a função de alocar determinadas requisições de clientes para as máquinas com a menor taxa de utilização para que com o compartilhamento de recurso nenhum cliente fique com indisponibilidade do serviço devido a alta utilização de recursos, nesse modelo além do provimento e distribuição de atualização escalável temos também uma escalabilidade no que se refere aos recursos.

O modelo nível 3 é muito utilizado por apresentar uma facilidade maior tanto na sua distribuição como na implantação no ambiente de produção, esse nível é de fácil implementação porém muito recurso é desperdiçado tendo em vista que será necessário que a instância no qual o software esteja inserida tenha um alto poder computacional para que a mesma possa processar todas as requisições mesmo em momentos de pico porém boa parte desse recurso é desperdiçado nos momentos de baixo pico podendo esse ser revertida quando se é implementado o nível de maturidade 4 onde todas as instâncias seriam contêineres e os mesmos seriam orquestrados para que só subissem com o aumento da demanda e fossem desligados de acordo com a diminuição das mesmas.

2.2 ARQUITETURA MULTI-TENANCY

A arquitetura multi-tenancy ou múltiplos inquilinos no português é a habilidade de lidar com os dados de múltiplos clientes em uma única instância da aplicação [1].

Para aplicações baseadas na nuvem a ideia de compartilhamento recursos é fundamental, principalmente para diminuição de custos e velocidade na implantação e entrega de novas atualizações.

No modelo arquitetural *multi-tenancy* utilizado para aplicações SaaS comumente se encontram no nível de maturidade 3 ou 4 como já foi comentado anteriormente, isso se dá por possuir a habilidade de manipular os dados de diferentes clientes através de uma mesma aplicação compartilhamento recursos, como é uma das características já presentes em ambos os níveis.

A ideia da arquitetura *multi-tenancy* é possibilitar para que através de uma mesma plataforma ou base de código todos os usuários possam gerenciar e manipular suas informações [1].

Existem algumas estratégias para isolamento de dados que serão abordadas mais profundamente nos próximos tópicos deste capítulo porém podemos separar as estratégias para isolamento dos dados em duas, as de isolamento total onde os dados são separados por bancos ou esquemas de bancos ou quando em um mesmo banco essa separação se dá por nome de tabela ou atributo identificador do cliente presente em cada tabela compartilhada, no primeiro caso é comum a utilização de middlewares para chaveamento de conexão seja ela com banco ou esquema de bancos, ambas as estratégias são muito utilizadas e para cada estratégia pontos positivos e negativos estão presentes, esses pontos poderão ser vistos com mais profundidade adiante visto que foram citados apenas para demonstrar que a de acordo com a estratégia adotada a abordagem a ser utilizada da arquitetura em questão para uma mesma aplicação poderá diferenciar e impactar tanto no curso do desenvolvimento quando no custo de manutenibilidade e escalabilidade dos mesmos.

2.3 ISOLAMENTO DE DADOS

Neste tópico serão abordados algumas das principais estratégias para isolamento de dados na arquitetura multi-tenancy, as estratégias podem se subdividir em duas, as que compartilham o mesmo banco e as que se utilizam de base de dados distintas, para isso pode-se analisar as estratégias de acordo com uma escala, das que possuem um maior isolamento para as que possuem um maior isolamento de dados conforme apresentado na Figura 2.

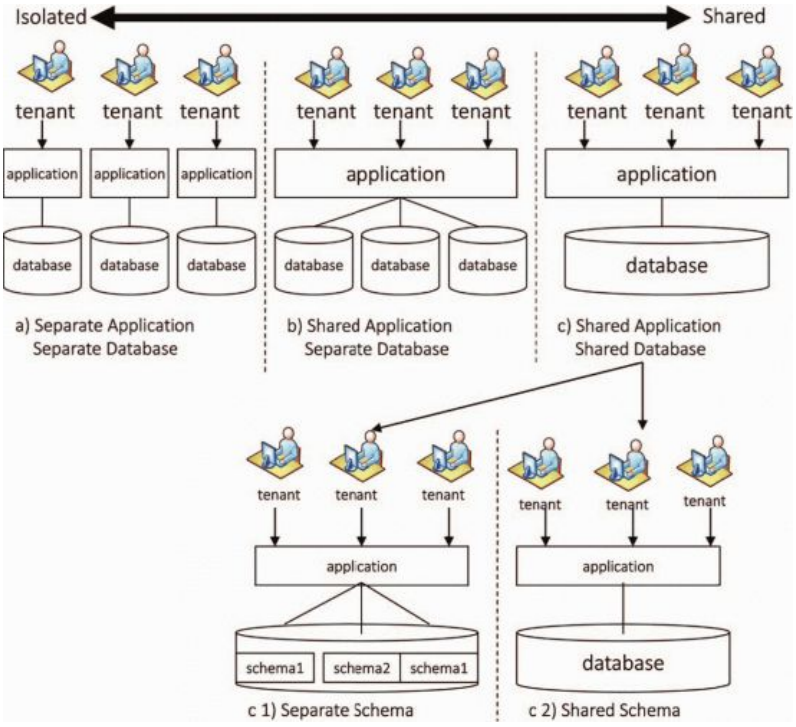


Figura 2 - Escala de isolamento de dados. [6]

Na Figura 2 pode-se notar algumas das estratégias para isolamento dos dados, algumas mais robustas e escaláveis e outras mais propícias a falha de segurança, logo abaixo irei explicar com maiores detalhes cada estratégia e seus pontos fortes e fracos, ao todo irei detalhar apenas duas estratégias que estão presentes na figura, a de um banco de dados por cliente e a de um banco único para todos, a figura apresenta outras estratégias porém que se encaixam nessa divisão.

Para a do banco de dados único temos duas possibilidades de implementação, podemos criar uma coluna em cada tabela que referencia um

identificador que seja único para cada cliente como mostra a Figura 3 ou a utilização de um prefixo no nome de cada tabela com o identificador, para essa estratégia teríamos várias tabelas com sua estrutura duplicada mudando apenas seu nome que teria como prefixo o identificador de cada cliente, para ambas as estratégias um dos principais problemas seria a escalabilidade, a criação de novas instâncias bem como a adição de novos clientes a base estariam limitadas a taxa de leitura e escrita do banco por se utilizarem da mesma instância.

Podemos verificar também que erros no processo de desenvolvimento que sejam repassados para produção poderão expor dados de outros clientes deixando visível que ambas as estratégias apesar de serem simples apresentam problemas no que se refere a segurança das informações [6].

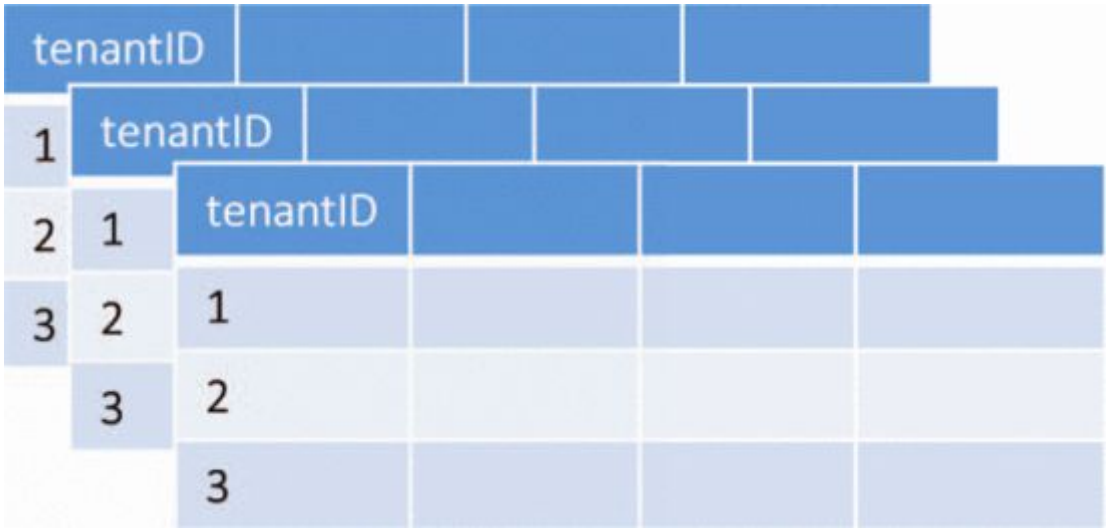


Figura 3 - Identificador do cliente por coluna. [6]

A última estratégia é de um único banco de dados para cada cliente, nessa estratégia para cada novo cliente a aplicação automaticamente cria um novo banco de dados em qualquer servidor de acordo com o nome do cliente, desse modo alguns requisitos são completamente cumpridos, quanto ao isolamento dos dados, esse modelo em comparação aos outros 2 apresentados é o que mais possui o isolamento adequado para segurança das informações ali contidas, em questão a escalabilidade nesse modelo um único cliente acessa um único banco, apesar do banco de dados continua possuindo a sua limitação de escrita e leitura porém essa limitação não será compartilhada com nenhum outro cliente já que cada um terá sua

base de dados separada e individual, a remoção ou até o backup pode ser feito de maneira simplificada e unitária já que cada um terá sua base individual.

Visto todas as estratégias adotou-se para este trabalho já visando a escalabilidade da aplicação a estratégia de base de dados compartilhadas visto que foi a que melhor apresentou pontos positivos na questão de isolamento dos dados pelo serviço que será construído por se tratar de informações sensíveis como logins e senhas que necessitam de uma atenção maior.

2.4 SEGURANÇA

A autenticação no contexto da computação é o processo no qual confirmamos a identidade de um usuários, para a confirmação de identidade, ou seja, de que a pessoa que está se utilizando das credenciais de um usuário específico é realmente o dono, os métodos vem cada vez mais se sofisticando porém a forma mais básica e utilizada é através de um login e senha, após realizado, o processo de identificação que consiste em verificar se existe algum usuário com aquele determinado *login* e se sua senha é a mesma que a informada, ele receberá a autorização do sistema ou serviço no qual ele irá se utilizar.

A autorização consiste em definir quais locais do sistema ou quais serviços a pessoa no qual efetuou o *login* está autorizada a se comunicar, essa autorização é comumente feita através de grupos de acessos que consistem em um aglomerado de informações utilizadas pelo próprio sistema que definem quais os locais o usuário é autorizado e para estes quais os acessos que o mesmo tem.

O acesso consiste em definir quais restrições o usuário autorizado tem sobre determinados recursos, essas restrições podem ser das mais variadas formas, como: restrição de *IP(Internet Protocol),* de tipo de operação, hora do dia no qual determinada operação pode ser realizada, ou seja, de acordo com o sistema ou serviço que venha a ser construído suas regras de acessos serão distintas uns dos outros, não tendo necessariamente uma padrão estabelecido.

2.5 CONSIDERAÇÕES FINAIS

Este capítulo apresentou os conceitos básicos sobre *Cloud Computing* e quais as características que as aplicações devem ter para serem chamadas de aplicações para nuvem bem como os seus níveis de maturidade. Logo após foi introduzido conceitos sobre a arquitetura multi-tenancy que é uma das arquiteturas utilizada para a construção desse tipo de aplicação bem como as estratégias utilizadas para o isolamento dos dados para esse tipo de arquitetura. Por fim alguns conceitos sobre Autenticação, Autorização e Acesso foram apresentados no âmbito mais geral a fim de apenas conceituar para que no capítulo posterior possa ser entendido como esses meios de segurança foram aplicados no serviço construído.

3 CONSTRUÇÃO DA APLICAÇÃO

Neste capítulo serão apresentados com maiores detalhes como os conceitos apresentados no capítulo anterior foram utilizados para a construção da solução, todo o código fonte da aplicação pode ser visto no repositório do Github[3].

3.1 FRAMEWORK

A aplicação foi construída utilizando uma framework MVC em Node.Js[4], a versão do Node utilizado foi a 10.14 e o framework utilizado foi o *AdonisJs*[5] na versão 4.1, apesar do framework utilizado apresentar a possibilidade de conexão com outros bancos de dados a aplicação foi construída utilizando como base o *PostgreSQL*[6] como banco de dados na versão 9.4.

Na versão atual a solução apresenta o sistema de autenticação utilizando os padrões do *JWT*[7] e para o gerenciamento de acesso e controle de usuários se implementou a abordagem de grupos e acessos.

3.2 ARQUITETURA

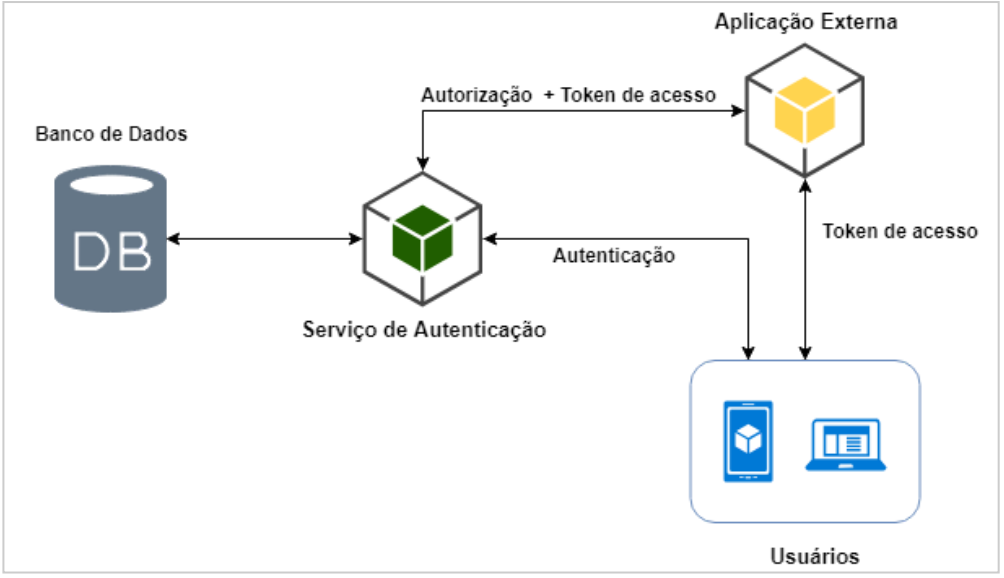


Figura 4 - Diagrama da arquitetura.

[3] https://github.com/jbsn94/multitenant-auth-service
[4] https://nodejs.org/en/
[5] https://adonisjs.com/
[6] https://www.postgresql.org/
[7] https://jwt.io/

A Figura 4 apresenta um como funcionará ao serviço construído, ao lado mais esquerdo pode-se visualizar o serviço de banco de dados onde constarão todas as informações de gerenciamento de autorização e acesso de cada usuários. Mais ao centro o serviço de autenticação construído que intermediará a comunicação com os usuários para a autenticação e irá fornecer informações de autorização a aplicações externas, o cubo em amarelo representando aplicações externas que se comunicam com o serviço bem como os usuários realizando a autenticação e comunicação com as aplicações que se utilizam desse serviço através da utilização do token de acesso fornecido após a autenticação.

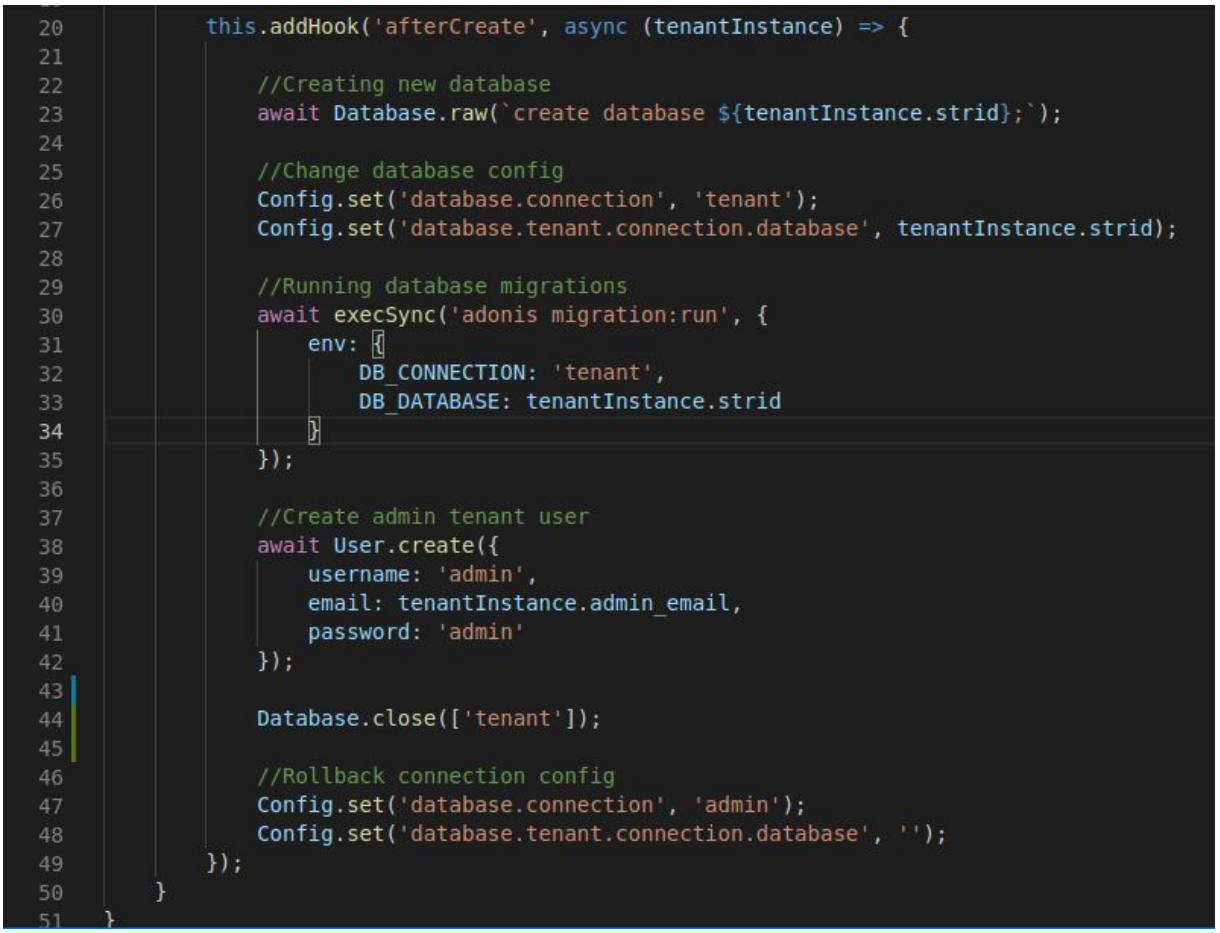
Nos tópicos a seguir serão detalhados como os conceitos abordados no capítulo 2 foram utilizados e implementados para a construção do serviço de autenticação, autorização e controle de acesso de usuários.

3.2.1 ISOLAMENTO DOS DADOS

Como ponto de partida para implementação da arquitetura foi necessário a implementação da estratégia de isolamento de dados para que pudesse ser viabilizado a utilização da arquitetura selecionada.

Criou-se um cadastro simples para os inquilinos onde era necessário informar o nome do cliente e um e-mail para que fosse criado um usuário administrador, ao se realizar esse registro na base administradora a aplicação executa a rotina de criação de um novo inquilino que consiste em criar uma nova base de dados a partir do nome do cliente adicionado de alguns caracteres aleatórios bem como criasse as tabelas bases para que esse novo banco pudesse ser utilizado e acessado pela aplicação como pode ser visto na Figura 5.

Com o isolamento de dados já implementado o passo seguinte se deu na implementação do multi-tenancy para que vários inquilinos possam se utilizar da mesma base de código para uma mesma instância da aplicação que será descrito no tópico seguinte com mais detalhes.



```
        this.addHook('afterCreate', async (tenantInstance) => {

            //Creating new database
            await Database.raw(`create database ${tenantInstance.strid};`);

            //Change database config
            Config.set('database.connection', 'tenant');
            Config.set('database.tenant.connection.database', tenantInstance.strid);

            //Running database migrations
            await execSync('adonis migration:run', {
                env: {
                    DB_CONNECTION: 'tenant',
                    DB_DATABASE: tenantInstance.strid
                }
            });

            //Create admin tenant user
            await User.create({
                username: 'admin',
                email: tenantInstance.admin_email,
                password: 'admin'
            });

            Database.close(['tenant']);

            //Rollback connection config
            Config.set('database.connection', 'admin');
            Config.set('database.tenant.connection.database', '');
        });
    }
}
```

Figura 5 - Rotina de criação de um novo inquilino

3.2.2 MULTI-TENANTS

Para viabilizar a utilização de múltiplos inquilinos foi necessário a criação de um *Middleware* que captasse todas as requisições que fossem direcionadas para um inquilino e modifica as configurações de conexão do banco para a do inquilino que estivesse recebendo a requisição.

Um *middleware* é uma classe que para o ciclo de vida de uma requisição no framework utilizado é a primeira a ser executada antes do controlador que é chamado posteriormente, é nele onde estão presentes os códigos que fazem a alteração da configuração de conexão com o banco de dados como pode ser visto na Figura 6.

Com a criação do *middleware* a arquitetura selecionada já está implementando, bastando apenas atrelar esse novo objeto a todas as rotas que são

referentes aos inquilinos, com isso tanto o isolamento dos dados quanto a arquitetura *multi-tenancy* com o nível de maturidade 3 já estariam habilitados na aplicação.

```
class DbSwitch {
  /**
   * @param {object} ctx
   * @param {Request} ctx.request
   * @param {Response} ctx.response
   * @param {Function} next
   */
  async handle ({ request, response }, next) {
    //Setting default connection
    Config.set('database.connection', 'admin');

    //Find tenant
    const tenant = await Tenant.findBy('name', request.params.clientesigla);

    //Check if tenant exists
    if(tenant) {

      //Change database connection configs
      Config.set('database.connection', 'tenant');
      Config.set('database.tenant.connection.database', tenant.$attributes.strid);

      //Pass request
      await next();

    } else {
      //Return error if tenant do not exists
      await response.status(404).json({
        message: 'Cliente not found'
      });
    }

  }
```

Figura 6 - Classe do middleware dbSwitch

3.2.3 SEGURANÇA

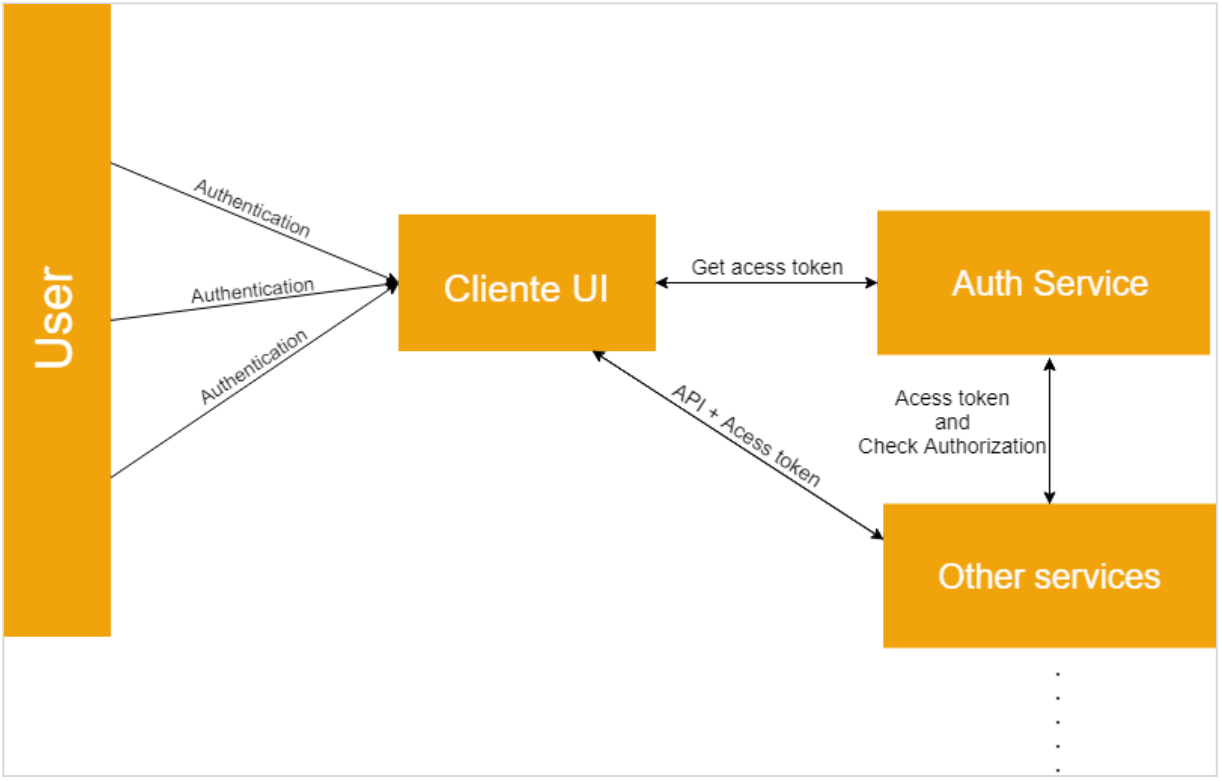


Figura 7 - Diagrama da arquitetura de segurança.

Nesta seção serão apresentadas as estratégias utilizadas para garantia de segurança da aplicação referente a autenticação, autorização e de controle e acesso implementados.

3.2.3.1 AUTENTICAÇÃO

Os padrões de autenticação aplicados no serviço construído é o mesmo para todos os inquilinos, adotou-se o padrão *JWT (Json Web Token)* como forma de transmissão segura de informações entre duas partes por apresentar sólidos padrões de seguranças muito bem documentados e reconhecidos [9].

Para se realizar a autenticação na aplicação algumas rotas específicas foram construídas únicamente para realização desse procedimento, para essas rotas é esperado que a aplicação que irá realizar a autenticação se utilizando deste serviço envie como parâmetro no corpo da requisição os atributos *login* e *password* com seus valores contendo e-mail e senha respectivamente, e ao se enviar será

retornado o *token jwt* que será reutilizado para que o usuário possa realizar outras operações de forma segura sem a necessidade de enviar constantemente seu e-mail e senha.

3.2.3.2 AUTORIZAÇÃO E CONTROLE DE ACESSO

Para a autorização e controle de acesso foram implementados os grupos, entidades e acessos, inicialmente na criação de um novo inquilino é criado como base a entidade *users* que se refere ao cadastro de usuários e para o mesmo são registrados os acessos de: *index*, *create, show, update e delete* bem como o grupo base administrador, a autorização e controle de acesso funciona da seguinte maneira, ao se criar um grupo será necessário enviar também as informações de quais acessos para determinadas entidades aquele grupo específico irá realizar, após realizado o registro do grupo o administrador poderá atrelar a quantos usuários forem necessários aos grupos de acessos para que eles possam realizar a operações necessárias, essas informações também serão enviadas através dos tokens gerados no login para informar quais os acessos e os grupos de um determinado usuário visto que qualquer alteração no corpo do token o invalidará automaticamente, visto que essa validação poderá ser realizada de forma automática através de uma rota específica.

3.3 CONSIDERAÇÕES FINAIS

Neste capítulo foram apresentados como os conceitos levantados no capítulo 2 foram utilizados para construção da solução e no capítulo seguinte serão apresentados os resultados obtidos com a aplicação desses conceitos.

4 DEMONSTRAÇÃO UTILIZANDO O SERVIÇO

Neste capítulo será demonstrado exemplos de uso do serviço construído. O *software* que está presente nas imagens para realização das requisições é o Postman[8].

Foram criadas duas seções, a primeira se refere a utilização dos serviços restritos ao gerenciamento de inquilinos e o segundo ao serviço de utilização dos inquilinos criados pelo gerenciador.

4.1 SERVIÇO DE GERENCIAMENTO DE INQUILINOS

Nesta seção será demonstrado a utilização dos serviços de gerenciamento de inquilinos presente na solução construída.

4.1.1 AUTENTICAÇÃO

Para a autenticação e recebimento do token de acesso o usuário administrador de todos os inquilinos irá realizar o login utilizando seu email e senha informado no corpo da requisição em formato *JSON* como demonstrado na Figura 8.

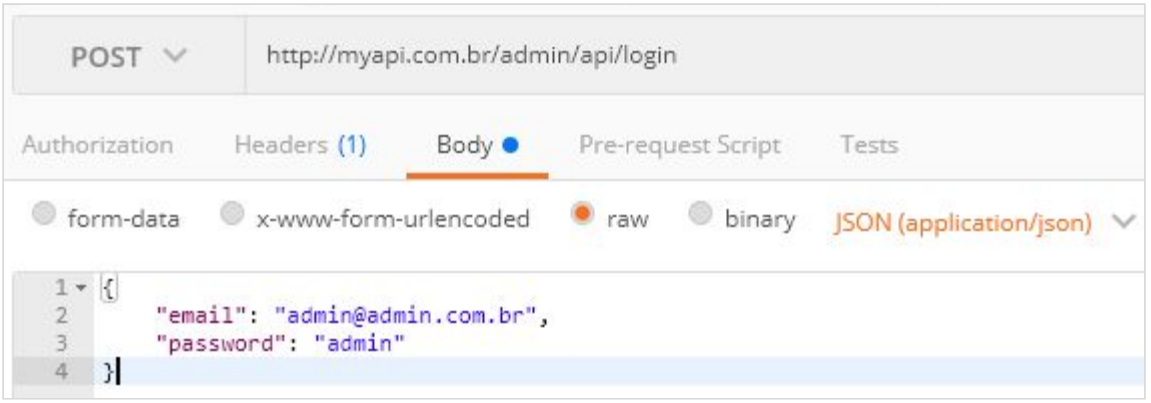


Figura 8 - Autenticação na API de gerenciamento de inquilinos

[8] https://www.getpostman.com/

Logo após a realização da autenticação será enviado um token para que o administrador possa, através dele, realizar todas as operações necessárias para gerenciamento dos inquilinos.

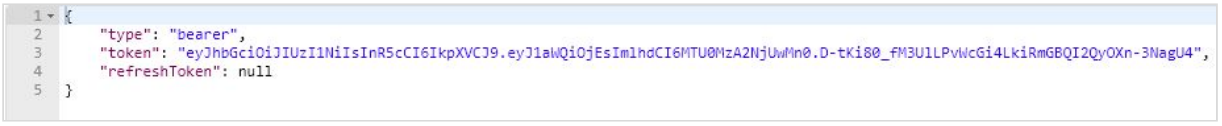


Figura 9 - Retorno da autenticação

4.1.2 CRIAÇÃO DE UM NOVO INQUILINO

Para criação de um novo inquilino será necessário enviar uma requisição *POST* que contenha em seu *Header* informando que o tipo do dado presente no corpo da requisição seja JSON, tal qual como foi realizado no passo anterior de autenticação e informe em também o valor para o atributo *Authorization* o valor *Barear* junto com o token recebido no passo de autenticação, será necessário separar o token da palavra *Barear* através de um espaço, e em seu corpo informar os valores de *name* e *admin_email* que são respectivamente o nome do novo inquilino e seu e-mail padrão de administrador, toda conta básica criada pelo sistema tem a senha padrão o *md5* do seu nome.

As Figuras 10 e 11 demonstram as informações aqui descritas como o Header e o corpo da requisição respectivamente e a Figura 12 mostra o retorno após realizado a criação do inquilino.

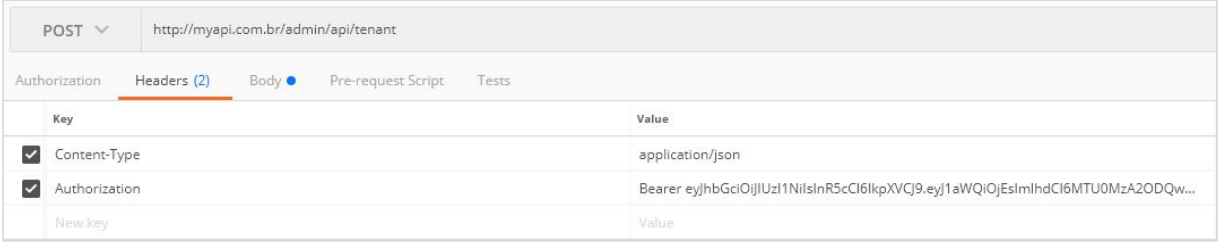


Figura 10 - Header da requisição de criação de um novo inquilino

Figura 11 - Corpo da requisição de criação de um novo inquilino

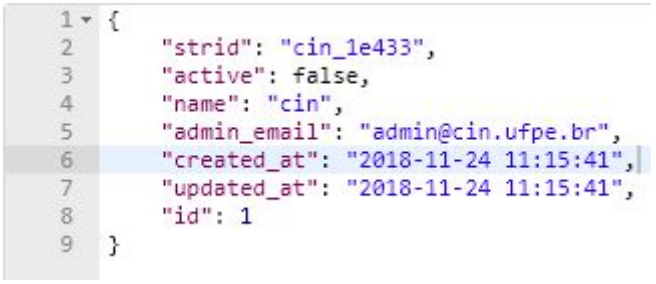


Figura 12 - Retorno JSON da requisição de criação de um inquilino

Como pode ser visualizado no retorno após a criação do inquilino é informado um atributo chamado *strid,* esse é o nome do banco de dados no qual o mesmo irá realizar a conexão, como mencionado na seção 3.2.1 para realização do isolamento dos dados, como pode ser visualizado na Figura 13.

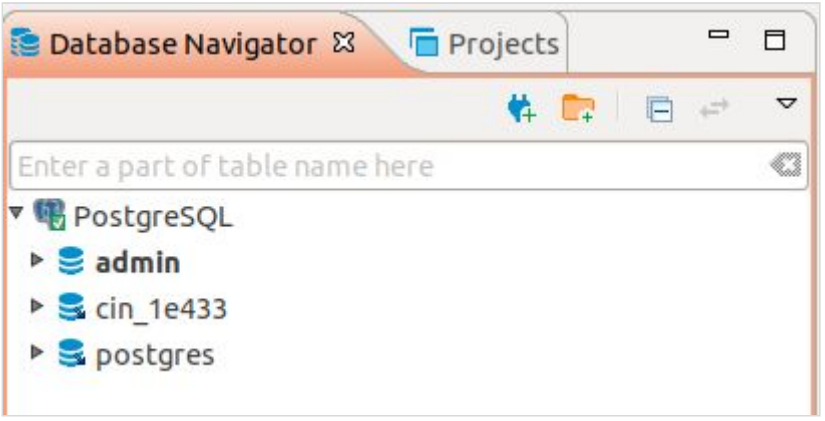


Figura 13 - Lista de banco de dados, visualização feita utilizando o DBeaver[9]

[9] https://dbeaver.io/download/

4.2 SERVIÇO DE AUTENTICAÇÃO DE INQUILINOS

O serviço de autenticação para inquilinos funciona da mesma forma como descrito na seção 4.1.1 porém a única diferença é na *URI*, ela está estruturada de maneira distinta para inquilinos e para os serviços dos administradores, as estruturas são:

- Administradores
    - admin/api/login
- Inquilinos
    - <*nomecliente*>/api/login

Para os inquilinos basta substituir na URI o valor de <*nomecliente*> pelo nome realizado no cadastro de inquilinos como demonstrado na seção 4.1.2.

4.3 CONSIDERAÇÕES FINAIS

Nesta seção foi apresentado o funcionamento do serviço construíndo contemplando os requisitos de isolamento de dados, autenticação, autorização e controle de acesso bem como na utilização da arquitetura *multi-tenancy* presente na aplicação.

5 CONCLUSÃO

Inicialmente foi realizado um estudo para levantamento das estratégias e soluções que seriam adotadas para a construção de uma aplicação de gerenciamento de segurança em uma arquitetura *multi-tenancy* tomando como critério de seleção além da segurança a escalabilidade e o isolamento dos dados.

Com isso este trabalho teve como seu principal objetivo a construção de um serviço de autenticação, autorização e gerenciamento de acesso e controle de usuários que além da autenticação dos usuários apresentasse como diferencial a partir das soluções existentes e mapeadas o gerenciamento e controle de acesso dos usuários para aplicações externas através da utilização dos tokens de acesso.

5.1 LIMITAÇÕES

Este trabalho se limitou a construção de uma aplicação em uma arquitetura *multi-tenancy* que fornece um serviço de autenticação, autorização e controle de acesso de usuários tendo como diferencial a criação de grupo e acessos para o gerenciamento e controle de acesso de usuários.

A aplicação construída fornece a autenticação de usuários utilizando sua credenciais bem como o fornecimento de suas informações de acesso através da utilização dos tokens de acesso para que aplicações externas possam realizar a verificações de segurança interna apropriadas.

5.2 TRABALHOS FUTUROS

A partir dos resultados obtidos notou-se a necessidade de melhorar o formato atual no qual o gerenciamento de acesso e controle de acesso dos usuários foi construído bem como a adaptação da aplicação para que a mesma possa ser utilizada em um ambiente de gerenciamento de contêineres do *Docker Swarm*.

REFERÊNCIAS

GURUDATT KULKARNI. **Multi-tenant SaaS clound**. Disponível em <https://ieeexplore.ieee.org/document/6757684> acessado em 29 de Setembro de 2018.

ZEESHAN PERVEZ, SUNGYOUNG LEE, YOUNG-KOO LEE. **Multi-Tenant, Secure, Load Disseminated SaaS Architecture.** Disponível em <https://ieeexplore.ieee.org/document/5440474> acessado em 29 de Setembro de 2018.

FARZANA SHAIKH, DIPTI PATIL. **Multi-tenant e-commerce based on SaaS model to minimize IT cost.** Disponível em <https://ieeexplore.ieee.org/document/7012861> acessado em 29 de Setembro de 2018.

National Institute of Standards and Technology. **The NIST Definition of Cloud Computing.** Disponível em <https://csrc.nist.gov/publications/detail/sp/800-145/final> acessado em 12 de Novembro de 2018.

Weiping Li, Zhichao Zhang, Si Wu, Zhonghai Wu. **An Implementation of the SaaS Level-3 Maturity Model for an Educational Credit Bank Information System.** Disponível em <https://ieeexplore.ieee.org/document/5494299> acessado em 20 de Setembro de 2018.

Huixin Chen. **Architecture strategies and data models of Software as a Service: A review.** Disponível em <https://ieeexplore.ieee.org/document/7586486> acessado em 24 de Setembro de 2018.

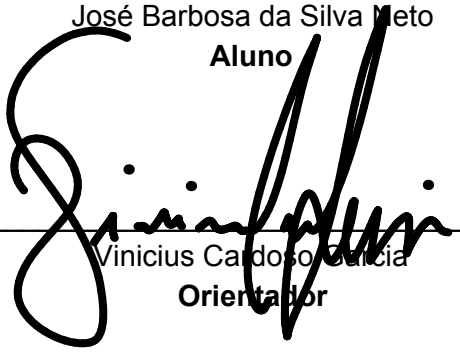
F. CHONG, G. CARRARO. **Architecture Strategies for Catching the Long Tail.** Frederick Chong e Gianpaolo Carraro, 2006.

AdonisJs. **AdonisJs - Node.js web framework.** Disponível em <https://adonisjs.com/> acessado em 20 de Novembro de 2018.

JSON Web Token. **Introduction to JSON Web Tokens.** Disponível em <https://jwt.io/introduction/> acessado em 18 de Novembro de 2018.

Assinaturas

José Barbosa da Silva Neto
Aluno



Vinicius Cardoso Garcia
Orientador