Instituto Tecnológico de Aeronáutica
Curso de Engenharia da Computação
Disciplina de CES-29 Engenharia de Software

Documentação Técnica

Relatório do Projeto Gislene

Dylan Nakandakari Sugimoto
Gabriel Adriano de Melo

Professor Dr. Inaldo Capistrano Costa

São José dos Campos
28/05/2018

Índice

Introdução

O método SCRUM prega por evitar a documentação e incentiva a comunicação informal. Contudo, documentar torna-se essencial para o projeto quando seu planejamento prevê uma sucessão de equipes. A documentação torna-se, portanto, uma necessidade vital para que o projeto continue a ser desenvolvido de forma ágil pela equipe seguinte. Assim, é responsabilidade da equipe anterior documentar as informações necessárias para a compreensão do que foi feito, de como funciona e do que se deve fazer. Dessa forma, como o projeto Gislene está nessa situação, este relatório é a documentação técnica que tem por objetivo explicar a estrutura do GIT, justificar a escolha das ferramentas de configuração *Bundler* e *Vagrant* e fornecer um guia de configuração possibilitando a execução do mesmo. Por fim, essa documentação foi validada pela equipe do Centro de Computação da Aeronáutica de São José dos Campos (CCA – SJC).

Documentação GIT

Escolheu-se utilizar o *GitHub* como plataforma de hospedagem de código e controle de versionamento utilizando o git e hospedou-se todo código do projeto em [1]. Na página principal (Figura 1), há a pasta do projeto, “projeto_padrao_OSM”, que é a pasta de um projeto criado pela ferramenta *RubyOnRails*. Isso significa que a organização do código dentro dessa pasta, “projeto_padrao_OSM”, segue a organização padrão do *RubyOnRails*. Além disso, nessa página, há o arquivo “.gitignore” configurado para ignorar os arquivos de sistema de um *commit* de um membro e os arquivos “README.md”, que é o guia da configuração do projeto, e “bancoDados.md”, que contém a configuração manual do banco de dados.

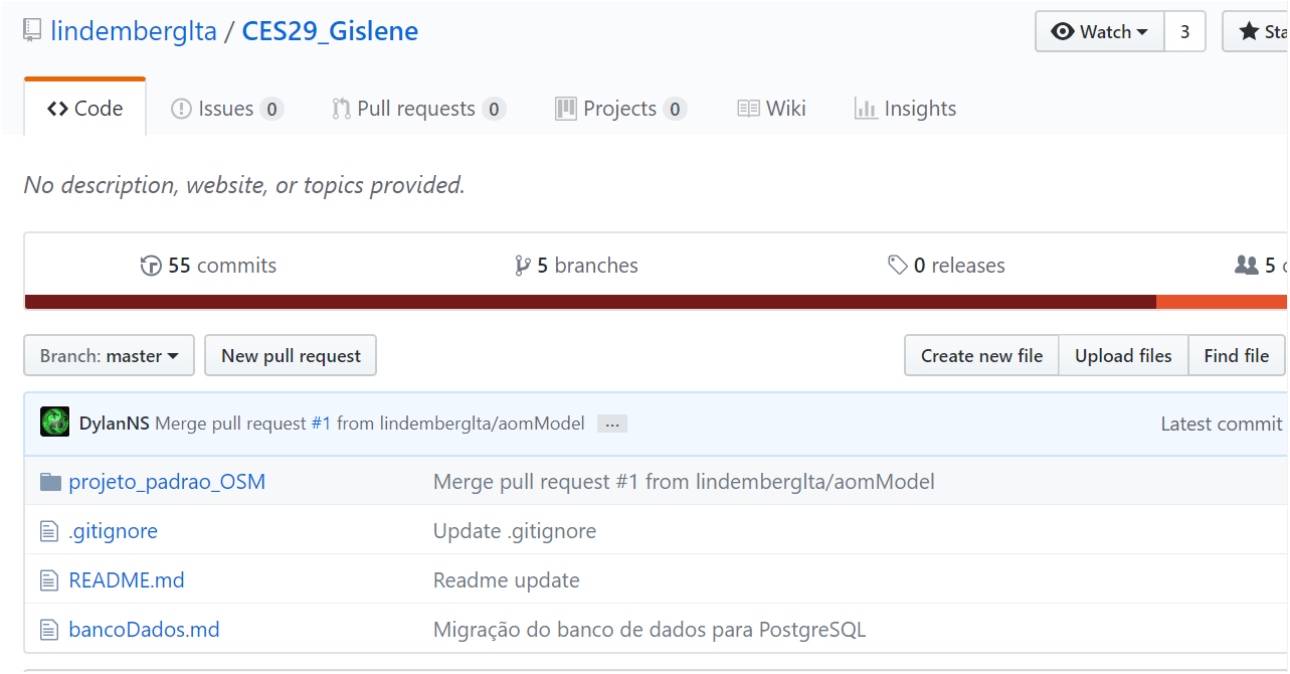


Figura 1: página inicial do GitHub [1].

Como dito anteriormente, a pasta “projeto_padrao_OSM” (Figura 2) está organizada seguindo o padrão de organização do *RubyOnRails* que é o seguinte [3]:

app/ – Dentro dessa pasta estão os arquivos da estrutura MVC – (Model, View e Controller);
config/ – parâmetros de configurações tanto do projeto quanto do banco de dados;
db/ – Informações sobre o esquema e as *migrations* geradas para modificar o banco de dados;
lib/ – Código biblioteca compartilhado, códigos que não podemos adicionar a uma um modelo, *view* ou *controller*, algo como um gerador de arquivos pdf. Não colocar biblioteca de vendor.

public/ – Diretório acessível pela Web. É a pasta onde o HTTP *server* utiliza pra rodar o seu aplicativo;

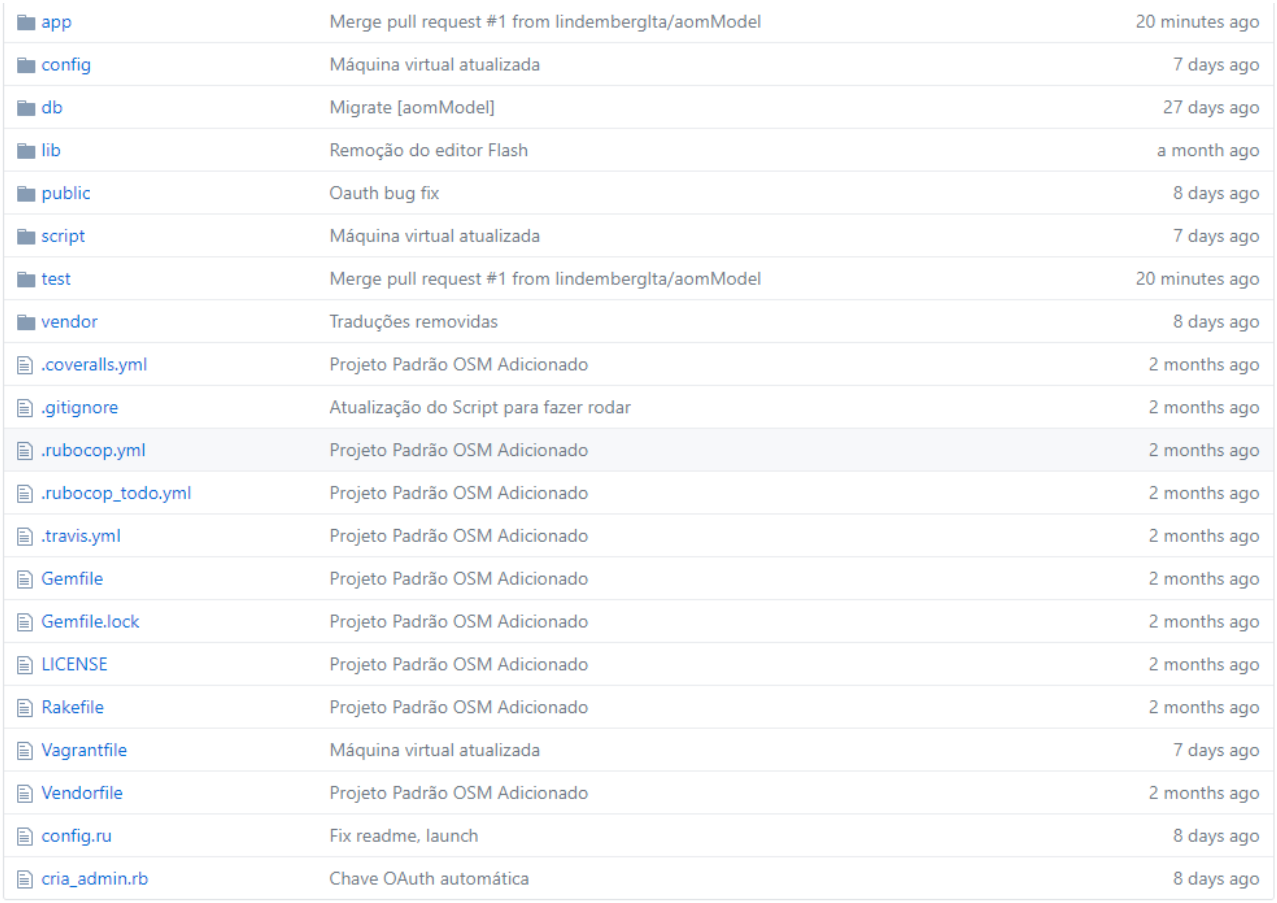
script/ – *Scripts* utilitários, onde estão os programas utilizados pelos desenvolvedores *Rails*, para entendê-los você pode executá-los sem nenhum argumento, você obterá informações sobre de uso;

test/ – Utilitários de testes, onde escrevemos os testes funcionais, testes de integração, *fixtures* e simulações;

vendor/ – Código importado onde você irá pôr os seus *plugins* utilizados pela aplicação, código de terceiros.

Rakefile – *script* de construção;

Gemfile/ - arquivo que descreve as dependências gem do projeto.[4]



app	Merge pull request #1 from lindembergIta/aomModel	20 minutes ago
config	Máquina virtual atualizada	7 days ago
db	Migrate [aomModel]	27 days ago
lib	Remoção do editor Flash	a month ago
public	Oauth bug fix	8 days ago
script	Máquina virtual atualizada	7 days ago
test	Merge pull request #1 from lindembergIta/aomModel	20 minutes ago
vendor	Traduções removidas	8 days ago
.coveralls.yml	Projeto Padrão OSM Adicionado	2 months ago
.gitignore	Atualização do Script para fazer rodar	2 months ago
.rubocop.yml	Projeto Padrão OSM Adicionado	2 months ago
.rubocop_todo.yml	Projeto Padrão OSM Adicionado	2 months ago
.travis.yml	Projeto Padrão OSM Adicionado	2 months ago
Gemfile	Projeto Padrão OSM Adicionado	2 months ago
Gemfile.lock	Projeto Padrão OSM Adicionado	2 months ago
LICENSE	Projeto Padrão OSM Adicionado	2 months ago
Rakefile	Projeto Padrão OSM Adicionado	2 months ago
Vagrantfile	Máquina virtual atualizada	7 days ago
Vendorfile	Projeto Padrão OSM Adicionado	2 months ago
config.ru	Fix readme, launch	8 days ago
cria_admin.rb	Chave OAuth automática	8 days ago

Figura 2: página da pasta do projeto [2].

Ferramenta de Configuração

Escolha

Escolheu-se o *Bundler* como gerenciador de dependências e o *Vagrant* como gerenciador de ambiente de trabalho.

Descrição

O *Bundler* gerencia o ambiente do *Ruby* de forma consistente para os projetos *Rails*, pesquisando e instalando as *Gems* nas versões exatas em que são necessárias. O *Bundler* é responsável por garantir que as dependências estejam atualizadas e assegura que as *Gems* cruciais estejam presentes em todos os três ambientes: de desenvolvimento, de teste e de produção [5]. As *Gems* são os pacotes, isto é, bibliotecas do *Ruby*, instaladas pelo gerenciador de pacotes *RubyGems* que é orquestrado pelo gerenciador de dependências *Bundler*.

O gerenciador de ambiente de trabalho é, na realidade, um gerenciador de máquina virtual, que instala uma máquina virtual e todas as suas dependências de outros programas dentro da mesma. O *Vagrant* é o programa responsável pela realização dessas tarefas, possibilitando um ambiente de trabalho unificado. [6] Essas configurações estão descritas no arquivo *Vagrantfile* e no *script* de execução provision.sh .

Assim, há duas camadas de gerenciamento de dependências: a camada mais interna responsável pelo gerenciamento de bibliotecas do *Ruby*, e a camada mais externa responsável pelo ambiente de trabalho, gerenciando o banco de dados e a instalação de programas necessários para a execução do projeto.

Justificativa

Pela escolha da linguagem do projeto ter sido *Ruby*, o *framework* de desenvolvimento *web*, que é o *Rails*, utiliza obrigatoriamente o *Bundler* para gerir todas as outras bibliotecas do *Ruby* necessárias para a execução do projeto. Como *Ruby* é uma linguagem interpretada, não há a geração de um arquivo executável próprio de uma *build*, mas a execução do código interpretado em seu ambiente *Ruby* [5]. Assim, o objetivo do *Bundler* é facilitar e automatizar o processo de execução do código Ruby pela gestão das bibliotecas instaladas.

O *Vagrant* foi escolhido por já ser utilizado no Projeto OSM como gerenciador do ambiente de trabalho de desenvolvimento [7]. Foram necessárias mudanças nas configurações

presentes no projeto original para que se utilizasse a versão mais recente de máquina virtual configurada disponível, utilizando-se o sistema operacional Ubuntu 16.04 de longo suporte. O *Vagrant*, produzido pela *HashiCorp*, oferece um fluxo de trabalho uniforme e simplificado, facilitando a execução do projeto em diferentes sistemas operacionais. Ele utiliza um arquivo de configuração declarativo que descreve todos os requisitos de software, pacotes, configuração do sistema operacional, usuários entre outras configurações de sistema. [6]

Guia de Configuração

O arquivo “README.md” em [1] é o guia de configuração do projeto. Esse arquivo orienta detalhadamente o usuário desenvolvedor para configurar uma máquina virtual com o projeto configurado pronto para rodar o servidor do projeto. Assim, para configurar o projeto é necessário antes fazer o download do *VirtualBox*, que é utilizado pela ferramenta *Vagrant* para criar uma máquina virtual onde vai rodar o servidor do projeto.

O *download* desses programas pode ser realizado nos *links* abaixo, respectivamente para o *Vagrant* e para o *Virtual Box*:

https://www.vagrantup.com/downloads.html

https://www.virtualbox.org/wiki/Downloads

Para usuários Linux, recomenda-se também utilizar o instalador dos *sites* acima para obter a versão mais recente, embora seja possível instalar tais programas com os comandos abaixo:

```
sudo apt-get install vagrant
sudo apt-get install virtualbox
```

Com os programas acima instalados e capazes de executar corretamente, é necessário ir até a pasta do projeto e iniciar a máquina virtual. A máquina virtual tem uma pasta sincronizada com a pasta do projeto. Assim, arquivos modificados nessa pasta refletem na máquina virtual. No *Windows* essa sincronização pode não funcionar, sendo necessário a edição dentro da própria máquina virtual.

Para baixar os arquivos do *Github*, digite no terminal o comando git clone, e para mudar para dentro da pasta do projeto, o comando cd.

```
git clone https://github.com/lindemberglta/CES29_Gislene
cd CES29_Gislene/projeto_padrao_OSM
```

Caso se esteja atrás de um *proxy*, é necessário configurá-lo na máquina virtual, para que a mesma possa acessar a internet para baixar os programas e as dependências necessárias. A ideia é declarar variáveis de ambiente com a autenticação para o *proxy*, instalar o *plugin* do *Vagrant* para lidar com o *proxy* e configurar mais variáveis de ambiente para serem utilizadas pelo *plugin*, que efetivamente coordenará o tráfego de rede da máquina virtual. Os comandos para os sistemas *Linux* são (lembrando de alterar o *user*, *password*, *host* e *port* do comando):

```
export http_proxy="http://user:password@host:port"
export https_proxy="http://user:password@host:port"
vagrant plugin install vagrant-proxyconf
```

E depois de instalado o *plugin* do *proxy* através do comando:

```
export VAGRANT_HTTP_PROXY="http://user:password@host:port"
export VAGRANT_NO_PROXY="127.0.0.1"
vagrant up
```

Caso o ambiente seja *Windows*, a única diferença são nas variáveis de ambiente:

```
set http_proxy=http://user:password@host:port
set https_proxy=%http_proxy%
vagrant plugin install vagrant-proxyconf
```

E depois de instalado o *plugin* do *proxy* através do comando:

```
set VAGRANT_HTTP_PROXY="%http_proxy%"
set VAGRANT_NO_PROXY="127.0.0.1"
vagrant up
```

A execução da máquina virtual, dentro da pasta do projeto que contém o *Vagrantfile*, se dá por meio do seguinte comando.

```
vagrant up
```

As Figuras 3, 4, 5 e 6 ilustram a saída desse comando, responsável por instalar e configurar a máquina virtual, baixando todas as dependências.

Se houver algum problema durante a instalação, a máquina deve ser parada com o comando de parada, inicializada novamente, e reconfigurada com o comando *vagrant provision*, que executará novamente o *script* de configuração inicial. É importante a reconfiguração da máquina caso tenha ocorrido algum erro na tentativa inicial.

```
vagrant halt
vagrant up
vagrant provision
```

```
gabriel@notega:~/Projetos/teste2$ git clone --depth=1 https://github.com/lindemberglta/CES29_Gislene
Cloning into 'CES29_Gislene'...
remote: Counting objects: 1330, done.
remote: Compressing objects: 100% (1210/1210), done.
remote: Total 1330 (delta 104), reused 1231 (delta 88), pack-reused 0
Receiving objects: 100% (1330/1330), 5.69 MiB | 734.00 KiB/s, done.
Resolving deltas: 100% (104/104), done.
gabriel@notega:~/Projetos/teste2$ cd CES29_Gislene/
gabriel@notega:~/Projetos/teste2/CES29_Gislene$ ls
bancoDados.md  projeto_padrao_OSM  README.md
gabriel@notega:~/Projetos/teste2/CES29_Gislene$ cd projeto_padrao_OSM/
gabriel@notega:~/Projetos/teste2/CES29_Gislene/projeto_padrao_OSM$ ls
app     config.ru      db       Gemfile.lock  LICENSE  Rakefile  test        vendor
config  cria_admin.rb  Gemfile  lib           public   script    Vagrantfile  Vendorfile
gabriel@notega:~/Projetos/teste2/CES29_Gislene/projeto_padrao_OSM$ vagrant up
Bringing machine 'default' up with 'virtualbox' provider...
==> default: Importing base box 'ubuntu/xenial64'...
==> default: Matching MAC address for NAT networking...
==> default: Checking if box 'ubuntu/xenial64' is up to date...
==> default: Setting the name of the VM: projeto_padrao_OSM_default_1527812879968_79142
==> default: Clearing any previously set network interfaces...
==> default: Preparing network interfaces based on configuration...
    default: Adapter 1: nat
==> default: Forwarding ports...
    default: 3000 (guest) => 3000 (host) (adapter 1)
    default: 5432 (guest) => 5433 (host) (adapter 1)
    default: 22 (guest) => 2222 (host) (adapter 1)
==> default: Running 'pre-boot' VM customizations...
==> default: Booting VM...
==> default: Waiting for machine to boot. This may take a few minutes...
    default: SSH address: 127.0.0.1:2222
    default: SSH username: vagrant
    default: SSH auth method: private key
    default: Warning: Connection reset. Retrying...
    default: Warning: Remote connection disconnect. Retrying...
    default: Warning: Connection reset. Retrying...
    default: Warning: Remote connection disconnect. Retrying...
    default: Warning: Connection reset. Retrying...
    default: Warning: Remote connection disconnect. Retrying...
    default: Warning: Connection reset. Retrying...
    default: Warning: Remote connection disconnect. Retrying...
    default:
    default: Vagrant insecure key detected. Vagrant will automatically replace
    default: this with a newly generated keypair for better security.
    default:
    default: Inserting generated public key within guest...
    default: Removing insecure key from the guest if it's present...
    default: Key inserted! Disconnecting and reconnecting using new SSH key...
==> default: Machine booted and ready!
==> default: Checking for guest additions in VM...
    default: The guest additions on this VM do not match the installed version of
    default: VirtualBox! In most cases this is fine, but in rare cases it can
```

Figura 3: Saída inicial para o comando de instalação da máquina virtual.

Executa-se automaticamente o *script* de instalação provision.sh, como indicado na Figura 4. Esse comando vai baixar a imagem de uma nova máquina virtual e instalar todas as dependências (inclusive o banco de dados) na mesma, o que pode demorar um pouco pela primeira vez. Observa-se também que a pasta compartilhada deve ser automaticamente montada.

```
    default: VirtualBox Version: 5.2
==> default: Mounting shared folders...
    default: /srv/projeto => /home/gabriel/Projetos/teste2/CES29_Gislene/projeto_padrao_OSM
==> default: Running provisioner: shell...
    default: Running: /tmp/vagrant-shell20180531-13106-6c5nb3.sh
    default: Generating locales (this might take a while)...
    default:   en_GB.UTF-8
    default: ...
    default:  done
    default: Generation complete.
    default: /tmp/vagrant-shell: line 10: warning: setlocale: LC_ALL: cannot change locale (en_GB.utf8)
    default: Hit:1 http://archive.ubuntu.com/ubuntu xenial InRelease
    default: Get:2 http://security.ubuntu.com/ubuntu xenial-security InRelease [107 kB]
    default: Get:3 http://archive.ubuntu.com/ubuntu xenial-updates InRelease [109 kB]
```

Figura 4: A inicialização correta da máquina monta a pasta compartilhada.

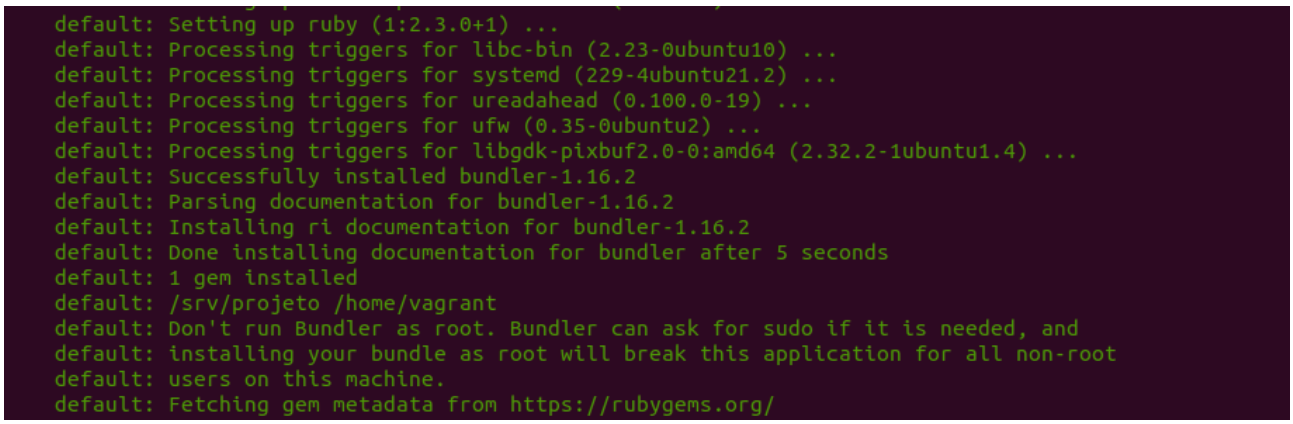


Figura 5: Instalação automática e execução do Bundler.

É criado automaticamente uma conta de administrador com uma aplicação *OAuth* para a autenticação da *API*. Esse código é executado automaticamente e pode ser visto no arquivo cria_admin.rb, que fica na pasta do projeto (último arquivo da Figura 2). Esse é o último *script* a ser executado na configuração inicial da máquina virtual, sendo a sua saída observada na Figura 6.

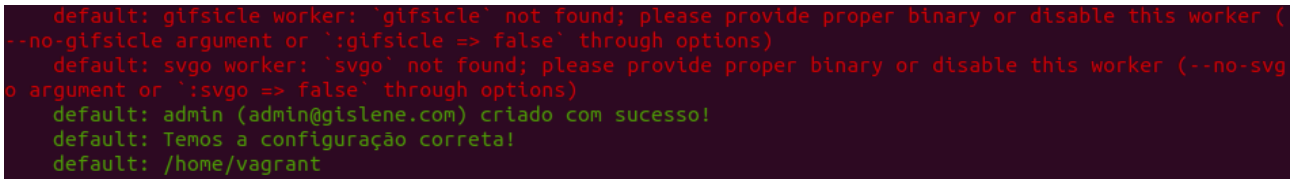


Figura 6: Finalização do *script* com sucesso, com apenas alguns avisos.

Assim que a instalação for concluída, pode-se logar na máquina virtual por meio do comando abaixo com saída observada na Figura 7 (comando dado na primeira linha dessa figura).

```
vagrant ssh
```

Dentro da máquina virtual existe uma pasta compartilhada do projeto, para acessá-la utilize o comando:

```
cd /srv/projeto/
```

No Windows, caso o compartilhamento de pastas não funcione corretamente, isto é, caso apareça um erro na página *web* após dar o comando para rodar o servidor do projeto, é necessário executar o arquivo *bash ClonarGit*.sh na pasta /home/vagrant. Esse *script* efetuará o *download* local do repositório e fará as inicializações necessárias. Os comandos para efetuar esses passos estão descritos abaixo:

```
cd /home/vagrant
bash ClonarGit.sh
cd /home/vagrant/CES29_Gislene/projeto_padrao_OSM
```

Assim o projeto deverá ser acessado apenas dentro da máquina virtual na pasta /home/vagrant/CES29_Gislene/projeto_padrao_OSM e não mais na /srv/projeto/.

```
gabriel@notega:~/Projetos/teste2/CES29_Gislene/projeto_padrao_OSM$ vagrant ssh
Welcome to Ubuntu 16.04.4 LTS (GNU/Linux 4.4.0-127-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

  Get cloud support with Ubuntu Advantage Cloud Guest:
    http://www.ubuntu.com/business/services/cloud

0 packages can be updated.
0 updates are security updates.


vagrant@ubuntu-xenial:~$ ls
clonarGit.sh
vagrant@ubuntu-xenial:~$ cd /srv/projeto/
vagrant@ubuntu-xenial:/srv/projeto$ ls
app            db             LICENSE   script                                  Vagrantfile
```

Figura 7: Acesso à máquina virtual pelo terminal.

Para executar o servidor, utilize o comando dentro da pasta do projeto:

```
rails server –binding=0.0.0.0
```

Qualquer modificação no código da pasta do projeto refletirá na máquina virtual, que usa a pasta compartilhada (/srv/projeto).

O servidor pode ser acessado localmente pelo navegador no endereço localhost:3000, e o banco de dados pode ser acessado pela porta 5433, com usuário *Vagrant* sem senha. Na página principal, pode-se criar uma conta e depois ir para a interface de Editar, indicada na Figura 8.

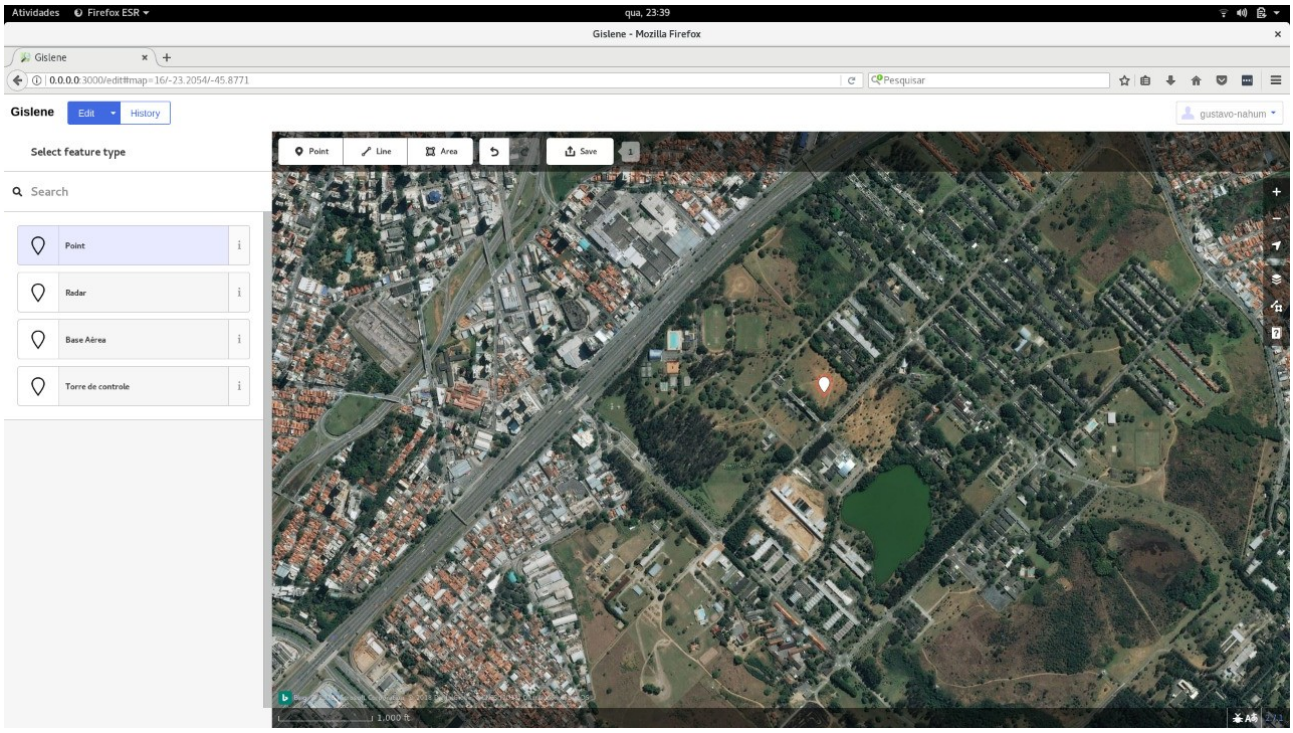


Figura 8: Captura de tela da interface de edição do mapa.

Comandos de Manutenção

Existem outros comandos auxiliares a serem executados no projeto *Rails*. Para rodar os testes existentes execute o seguinte comando na pasta do projeto:

```
bundle exec rake test
```

Para atualizar o banco de dados depois de criar um *model*:

```
bundle exec rake db:migrate
```

Teste de cobertura estáticos:

```
sudo gem install rcov
rcov -x gems test/*/*.rb
```

Para gerar a documentação automática do código:

```
rake doc:app
```

Permissão de *admin*:

```
$ bundle exec rails console
>> user = User.find_by_display_name("My New User Name")
=> #[ ... ]
>> user.roles.create(:role => "administrator", :granter_id => user.id)
=> #[ ... ]
>> user.roles.create(:role => "moderator", :granter_id => user.id)
=> #[ ... ]
>> user.save!
=> true
>> quit
```

O *Rails* faz um *log* automático, que pode ser visto em

```
tail -f log/development.log
```

Instalação Manual

Caso deseje-se executar o projeto manualmente, sem ter que instalar uma máquina virtual, pode-se seguir este outro guia. Todos os comandos citados abaixo estão disponíveis no arquivo de *script* provision.sh na pasta de *script* dentro de Vagrant/setub. Deve-se usar o Ubuntu 16.04 e dar os comandos para instalar os programas necessários (*Ruby*, bibliotecas de *ruby*, node.js, Postgre SQL, Apache, bibliotecas de imagem, *Git*, dentre outros):

```
sudo apt-get install ruby2.3 libruby2.3 ruby2.3-dev \
         libmagickwand-dev libxml2-dev libxslt1-dev nodejs \
         apache2 apache2-dev build-essential git-core \
         postgresql postgresql-contrib libpq-dev postgresql-server-dev-all \
         libsasl2-dev imagemagick libffi-dev
sudo gem2.3 install bundler
```

O *Bundler* é utilizado para gerir as dependências de código *ruby*. Dar o comando abaixo na pasta do projeto para instalar as dependências:

```
bundle install
```

É preciso configurar a config/application.yml. Para configurar basta dar o comando abaixo na pasta do projeto:

```
cp config/example2.application.yml config/application.yml
```

E também as configurações do banco de dados do projeto, dar o comando abaixo na pasta do projeto para configurar o banco de dados.

```
cp config/example.database.yml config/database.yml
```

Criar a conta no banco de dados com o seu nome de usuário

Um PostgreSQL role, precisa ser superusuário. Caso já existe um usuário do banco de dados criado com o mesmo nome do usuário do sistema, digite a senha do mesmo no arquivo config/database.yml

```
sudo -u postgres createuser --interactive
```

Criar os bancos de dados

Executar o *script* de criação do banco de dados

```
bundle exec rake db:create
```

PostgreSQL Btree-gist

É necessário habilitar a extensão btree-gist

```
psql -d gislene -c "CREATE EXTENSION btree_gist"
```

PostgreSQL Funções especiais

É necessário compilar as funções

```
cd db/functions
make libpgosm.so
cd ../..
```

E instalar as mesmas no banco de dados

```
psql -d gislene -c "CREATE FUNCTION maptile_for_point(int8, int8, int4) RETURNS
int4 AS ``pwd`/db/functions/libpgosm', 'maptile_for_point' LANGUAGE C STRICT"
psql -d gislene -c "CREATE FUNCTION tile_for_point(int4, int4) RETURNS int8 AS
``pwd`/db/functions/libpgosm', 'tile_for_point' LANGUAGE C STRICT"
psql -d gislene -c "CREATE FUNCTION xid_to_int4(xid) RETURNS int4 AS
``pwd`/db/functions/libpgosm', 'xid_to_int4' LANGUAGE C STRICT"
```

Estrutura do banco de dados

Criar as estruturas com o comando:

```
bundle exec rake db:migrate
```

Executar os testes

Para verificar que o projeto está funcionando corretamente

```
bundle exec rake test:db
```

Executar o servidor localmente

```
bundle exec rails server
```

Conclusão

Como forma de facilitar a acessibilidade do código fonte do projeto e o controle de versionamento, o *GitHub* foi, portanto, necessário e suficiente para suprir essa necessidade. Ao seguir o padrão de organização das pastas do *RubyOnRails*, tentou-se facilitar também a compreensão geral de onde está cada arquivo e qual a função de cada arquivo ou diretório. Além disso, aproveitou-se as ferramentas *Bundler* e *Vagrant* para resolver as dependências externas do projeto e realizar a configuração de uma máquina virtual, respectivamente. Dessa forma, combinadas as duas ferramentas, tem-se uma configuração automatizada do projeto, que é uma necessidade tendo em vista que uma outra equipe pode dar continuidade ao projeto. Portanto, utilizada uma ferramenta que facilite ter o projeto configurado e com todas as dependências resolvidas em uma nova máquina de forma automatizada é de fundamental importância para uma transição mais ágil e fluida. Por fim, essa documentação foi validada pela equipe do Centro de Computação da Aeronáutica de São José dos Campos (CCA - SJC).

Referências Bibliográficas

[1] Projeto Gislene. Disponível em: <https://github.com/lindemberglta/CES29_Gislene>. Acesso em 28 de maio de 2018.

[2] Projeto Gislene. Disponível em: <https://github.com/lindemberglta/CES29_Gislene/tree/master/projeto_padrao_OSM>. Acesso em 28 de maio de 2018.

[3] TOMIO, Fabio. Sobre a organização dos diretórios no RubyOnRails. Disponível em: <http://fabiotomio.com.br/blog/2009/01/17/entendo-a-estrutura-de-diretorios-do-rails/>. Acesso em 28 de maio de 2018.

[4] Tosbourn. Sobre GemFile. <Disponível em: https://tosbourn.com/what-is-the-gemfile/>. Acesso em 28 de maio de 2018.

[5] Contribuidores Bundler. Sobre o Bundler. Disponível em: <http://bundler.io/>. Acesso em 28 de maio de 2018.

[6] Hashicorp. Sobre o Vagrant. Disponível em: <http://www.vagrantup.com/>. Acesso em 28 de maio de 2018.

[7] Colaboradores do OpenStreetMap. Desenvolvimento do Projeto OpenStreetMap. Disponível em: <https://wiki.openstreetmap.org/wiki/Develop>. Acesso em 28 de maio de 2018.