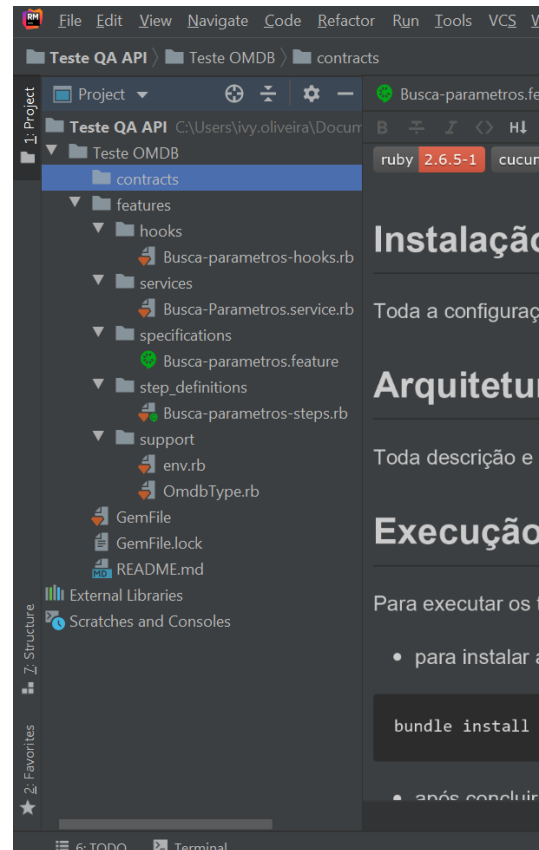


Visão Geral do projeto

- 1 – specifications
- 2 – step_definitions
- 3 – services
- 4 – support
- 5 – hooks
- 6 – reports



1. Specifications

Essa pasta contém o BDD do projeto

```
Busca-parametros.feature
1  #Language: pt
2  #utf-8
3
4  @validacao_paramiter
5  >> Funcionalidade: Validar os tipos de filmes
6
7  @validacao_type
8  >> Esquema do Cenário: Validar os tipos de filmes
9      Quando realizar a requisição informando o <type>
10     Então o sistema retorna o código <retorno>
11
12  Exemplos:
13      | type          | retorno |
14      | "movie"       | 200    |
15      | "series"      | 200    |
16      | "episode"     | 200    |
17      | "game"        | 200    |
18
```

2. Step Definitions

Nesta pasta contém as informações dos cenários

```
Busca-parametros-steps.rb x
1 Quando('realizar a requisição informando o {string}') do |type|
2   if (type == 'movie')
3     $type = $search_type_movie
4   elsif (type == 'series')
5     $type = $search_type_serie
6   elsif (type == 'episode')
7     $type = $search_type_episode
8   else
9     (type == 'game')
10    $type = $search_type_game
11  end
12
13  @post = @paramiter.search_paramiter_type
14 end
15
16 Quando('realizar a requisição informando o ano {float}') do |y|
17   if (y == 1962)
18     $y = $search_y_1962
19   elsif (y == 1971)
```

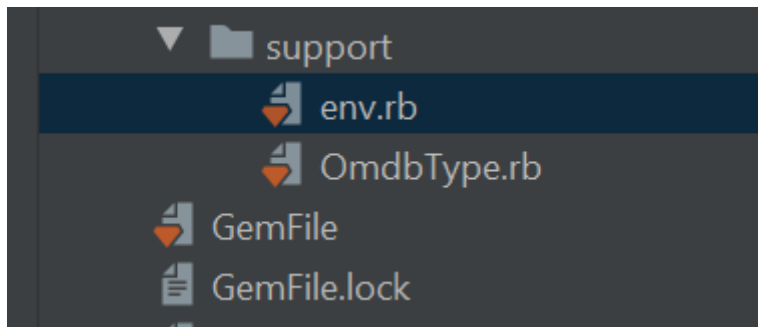
3. Services

Nesta pasta contém o código que com auxilio do HTTParty, realiza as requisições do teste, nela construímos as chamadas da API.

```
Busca-Parametros.service.rb x
1 #encoding: UTF-8
2
3 class ServiceOMDB
4   include HTTParty
5   # debug_output $stdout
6   base_uri uri 'http://www.omdbapi.com/'
7   format :json
8   headers "Content-Type" => "application/json;charset=utf-8"
9
10  def search_paramiter_type
11    self.class.get( path '', :query => { apiKey: "52ec71bf", s: "300", type: $type, y: $y, page: $page,
12                                          callback: $callback, v: $v, r: $r})
13  end
14 end
```

4. Support

Nesta pasta contém os arquivos relacionados ao suporte do projeto, lá dentro neste caso temos o arquivos: env.rb e o arquivo de massas.

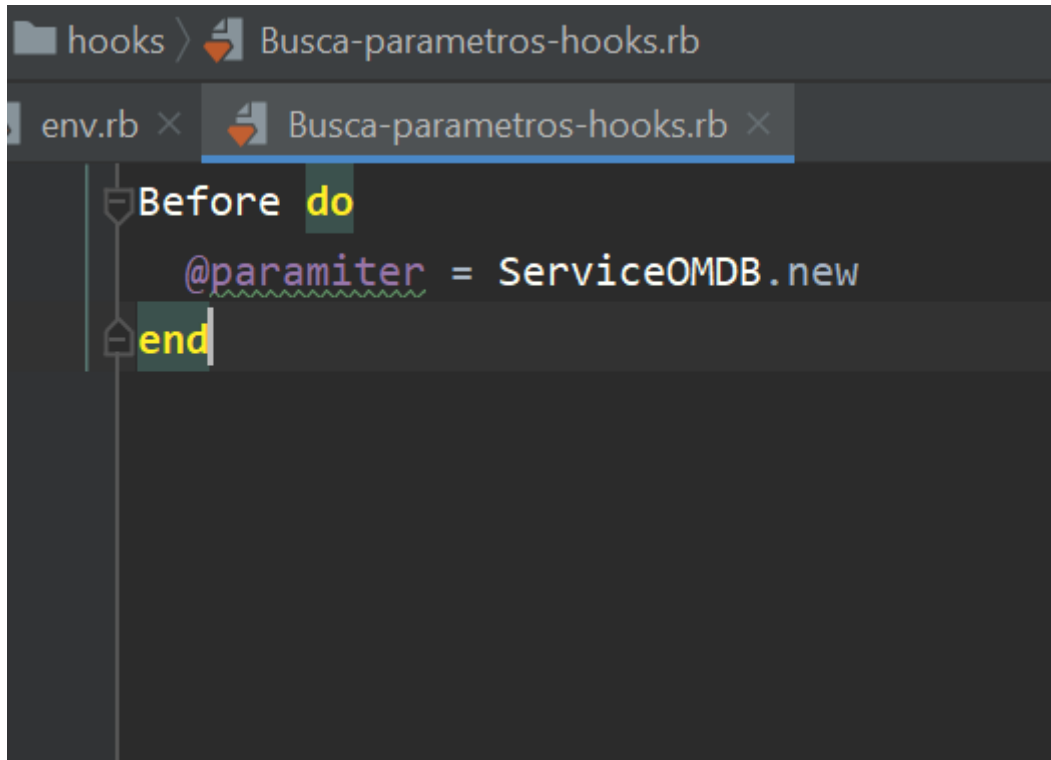


Arquivo env.rb

```
env.rb x
1  # -*- encoding : utf-8 -*-
2  require 'httparty'
3  require 'httparty/request'
4  require 'httparty/response/headers'
5  require 'rspec'
6  require 'allure-cucumber'
7  require 'cucumber'
8  require 'pry'
9  require 'test-unit'
10 require 'aspector'
11 require 'openssl'
12 require 'open-uri'
13 require 'faker'
14 require 'cpf_faker'
15
16 ENVIRONMENT = ENV['ENVIRONMENT']
17
18 Cucumber::Core::Test::Step.module_eval do
19   def name
20     return text if text == 'Before hook'
21     return text if text == 'After hook'
22     "#{source.last.keyword}#{text}"
23   end
end
```

5. Hooks

Nesta pasta contém o arquivo onde são executadas as ações necessárias antes e depois da execução dos testes.

A screenshot of a code editor interface. The top bar shows the file path 'hooks > Busca-parametros-hooks.rb'. Below the top bar, there are two tabs: 'env.rb' and 'Busca-parametros-hooks.rb'. The 'Busca-parametros-hooks.rb' tab is active and shows the following Ruby code:

```
Before do
  @paramiter = ServiceOMDB.new
end
```

The code is written in a dark-themed editor with syntax highlighting. The 'Before' keyword is in white, 'do' is in yellow, '@paramiter' is in purple, 'ServiceOMDB.new' is in white, and 'end' is in yellow. The variable '@paramiter' has a red squiggly line underneath it, indicating an undefined variable. The 'end' keyword has a small icon to its left.

6. Reports

Utilizo reports com visão web porque visualmente são mais estruturados.

Para gerar o relatório utilizamos os comandos:

Cucumber -p allure: executa dependência de relatórios

Allure serve [nome da pasta do projeto] – para que seja salvo no diretório de reports.