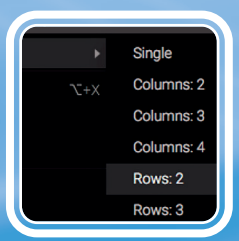
Programação Orientada a Serviços

Everton Fagner Costa de Almeida

Curso técnico nível médio subsequente em Informática para Internet



INSTITUTO FEDERAL
Rio Grande do Norte
Campus Avançado Natal - Zona Leste

editoraifrn

Didáticos

Programação Orientada a Serviços

Everton Fagner Costa de Almeida

Curso técnico nível médio subsequente em Informática para Internet

Instituto Federal de Educação, Ciência e Tecnologia do Rio Grande do Norte.

editoraifrn Didáticos

Natal-RN

2022

Presidente da República
Jair Messias Bolsonaro

Ministro da Educação
Victor Godoy Veiga

Secretário de Educação
Profissional e Tecnológica
Tomás Dias Sant´Ana

INSTITUTO FEDERAL
Rio Grande do Norte
Campus Avançado Natal - Zona Leste

Reitor
José Arnóbio de Araújo Filho

Pró-Reitor de Pesquisa e Inovação
Avelino Aldo de Lima Neto

Caderno elaborado em parceria entre o Instituto Federal de Educação, Ciência e Tecnologia e o Sistema Escola Técnica Aberta do Brasil – e-Tec Brasil.

Comitê Editorial da Diretoria de Educacão a Distância e Tecnologias Educacionais - Campus Avançado Natal Zona Leste/IFRN

Presidente
Wagner de Oliveira

Membros
José Roberto Oliveira dos Santos
Albérico Teixeira Canario de Souza
Glácio Gley Menezes de Souza
Wagner Ramos Campos

Suplentes
João Moreno Vilas Boas de Souza Silva
Allen Gardel Dantas de Luna
Josenildo Rufino da Costa
Leonardo dos Santos Feitoza

Equipe | Produção de Material Didático

Equipe de Elaboração
Cognitum

Coordenação Institucional
COTED

Projeto Gráfico
Eduardo Meneses e Fábio Brumana

Revisão linguística
Wagner Ramos Campos

Revisão tipográfica
Maria Valeska Rocha da Silva
Rodrigo Pessoa

Diagramação
Leonardo dos Santos Feitoza

Ficha catalográfica

A447p

Almeida, Everton Fagner Costa de.
Programação Orientada a Serviços. / Everton Fagner Costa de Almeida, -- 2022.
86 f. ; 30cm.

Caderno (Curso técnico nível médio subsequente em Informática para Internet). Instituto Federal de Educação, Ciência e Tecnologia do Rio Grande do Norte, Natal (RN), 2022.

ISBN: 978-65-84831-33-9

1. Educação 2. Caderno 3. Programação 4. Curso Técnico Subsequente I. Título.

CDU: 004.43

Catalogação na publicação pelo Bibliotecário-Documentalista
Ezequiel da Costa Soares Neto CRB15/613
Biblioteca Sebastião Názaro do Nascimento (BSNN) – IFRN

Sumário

Aula 1 - Conhecendo o ambiente de desenvolvimento - Parte 1

Olá, caro(a) cursista! Seja bem-vindo(a) à disciplina Programação Orientada a Serviços. Nesta Aula 1, vamos conhecer o Codeanywhere, uma plataforma de desenvolvimento on-line (PDO), que nos permite construir nossos projetos, sem a necessidade de instalação e de configuração de softwares. Aprenderemos como fazer uma conta gratuitamente; como criar um container usando uma stack; quais as funções do Codeanywhere; e como compartilhar um container.

Objetivos de aprendizagem

- Conhecer a plataforma de desenvolvimento Codeanywhere.
- Aprender a criar uma conta gratuitamente.
- Criar um container usando uma stack.

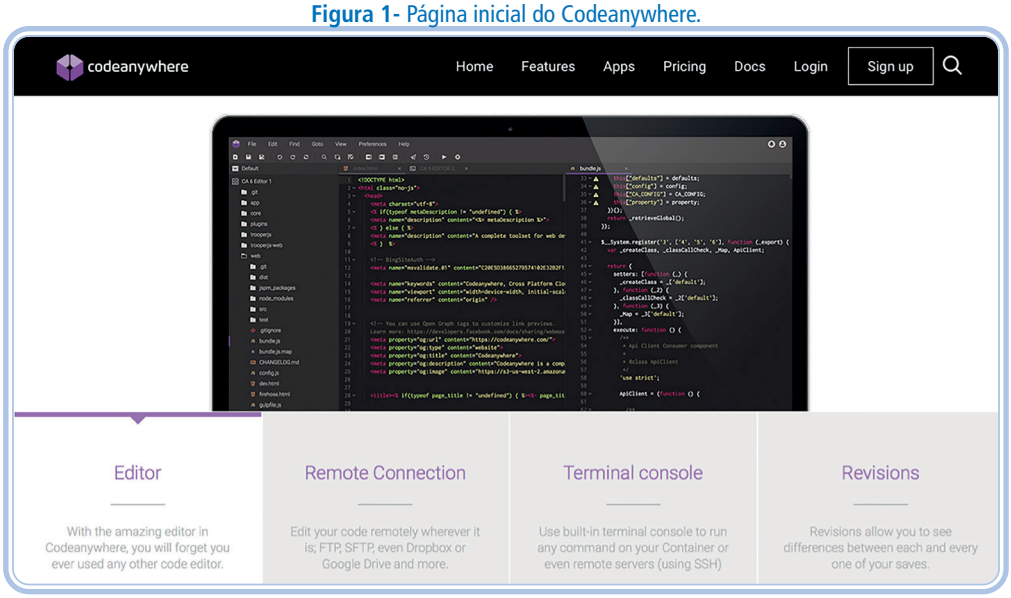
Desenvolvendo o conteúdo

Para que possamos desenvolver aplicações usando Ruby on Rails, inicialmente precisamos preparar o ambiente de desenvolvimento, que consiste na instalação e na configuração do sistema operacional e de outros softwares necessários, como o Ruby, o banco de dados, o sistema de controle de versões, o próprio framework Rails, o editor de código, entre outros.

Assim, a fim de facilitar esse processo, utilizaremos uma plataforma de desenvolvimento na nuvem, conhecida como Codeanywhere[1] , um ambiente de desenvolvimento on-line, o qual nos permite construir aplicações, a partir de diversas linguagens e frameworks, sem ser preciso configurar e instalar cada software necessário, pois tudo já estará pronto para ser usado. A seguir, a Figura 1 apresenta a página inicial do site do Codeanywhere.

1 Acesse essa plataforma: https://codeanywhere.com.

Figura 1- Página inicial do Codeanywhere.



Fonte: acervo pessoal.

O Codeanywhere está disponível através do seu navegador de internet (Google Chrome, Mozilla Firefox etc.), Android e iOS. Faremos o uso da versão Web dessa plataforma, contudo, você pode fazer o download das versões para smartphone.

1.1 Criando sua conta

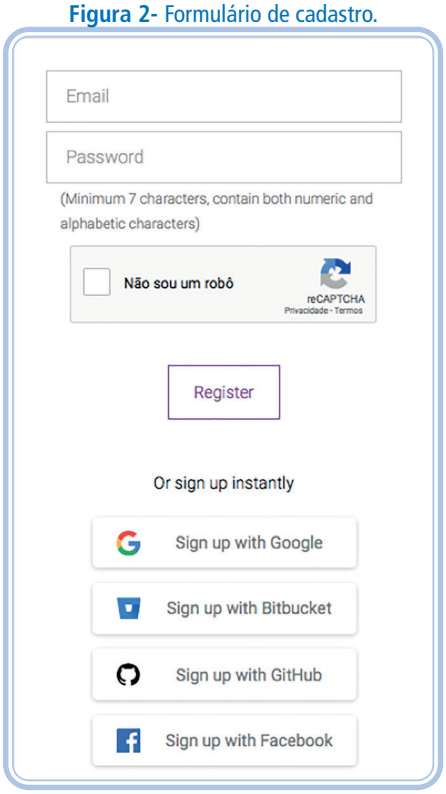
Para criar sua conta gratuitamente no Codeanywhere, siga os passos seguintes:

1- Acesse o endereço https://codeanywhere.com.

2- Clique no link Sign up, o qual está localizado no canto superior direito, como na Figura 1. Ou, simplesmente, acesse o site: https://codeanywhere.com/signup.

3- Feitos os procedimentos anteriores, você deverá visualizar o formulário de cadastro ilustrado na Figura 2.

Figura 2- Formulário de cadastro.



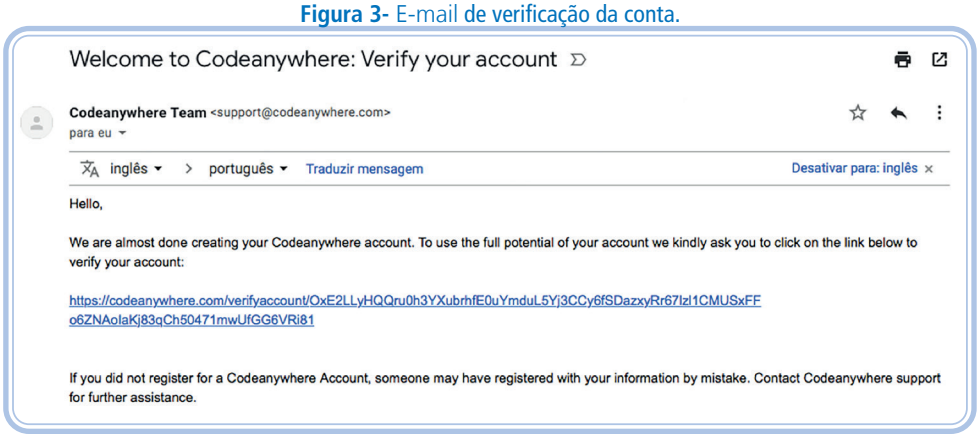
Fonte: acervo pessoal.

4- Nesse formulário, preencha o primeiro campo com o seu endereço de e-mail; o segundo campo, com uma senha.

5- Clique em "Não sou um robô" e, por fim, clique em Register.

Após a submissão desse formulário, sua conta no Codeanywhere estará criada. Mas ainda é preciso confirmar seu endereço de e-mail. Desse modo, verifique a caixa de entrada do endereço de e-mail, que você preencheu no formulário, e procure uma mensagem com o título "Welcome to Codeanywhere: Verify your account", conforme a Figura 3.

Figura 3- E-mail de verificação da conta.



Fonte: acervo pessoal.

Ao localizar essa mensagem, clique no link que aparece nela. Você será redirecionado para uma página, que irá confirmar seu endereço de e-mail. Caso não, localize essa mensagem, verificando a sua caixa de SPAM.

ATIVIDADE

1) Siga as orientações da aula para criar sua conta no Codeanywhere.

1.2 Criando um *container*

Para começar a usar o Codeanywhere, devemos criar um container. Você pode entender um container como um computador virtual, o qual te permite ligá-lo, desligá-lo, instalar um sistema operacional completo (geralmente Linux), entre outras funções típicas de um computador.

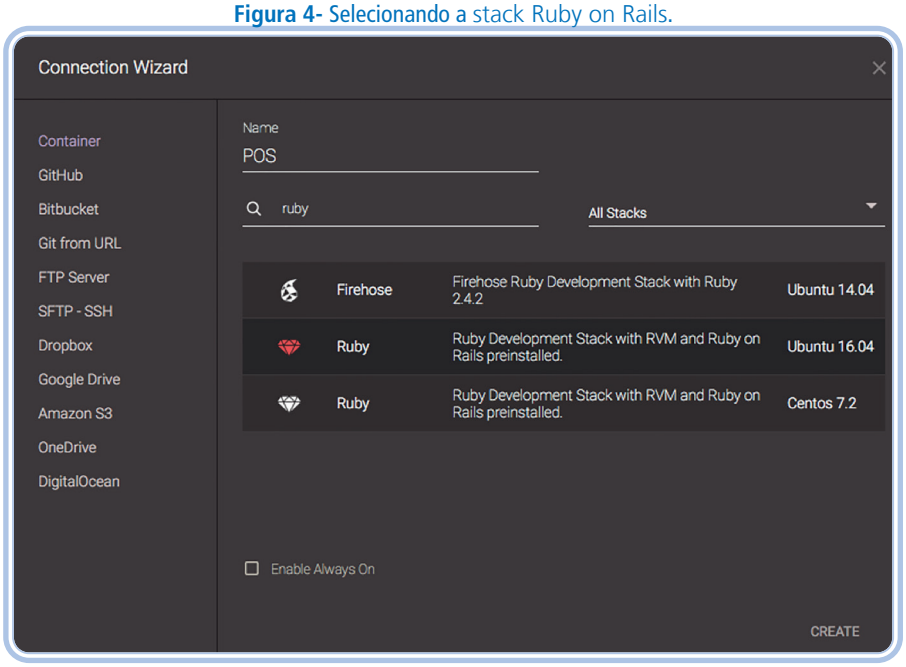
Ao criar um container, podemos escolher uma stack, que consiste em um sistema operacional específico com vários softwares já instalados. No nosso caso, instalaremos uma stack específica, com todos os softwares que precisamos para o desenvolvimento de aplicações em Ruby on Rails.

Assim, elencamos instruções para você criar seu primeiro container:

1- Acesse o Editor do Codeanywhere, através do endereço https://codeanywhere.com/editor.

2- Feito isso, o formulário ilustrado na Figura 4 deverá ser apresentado; caso não seja exibido na tela que abriu, clique no menu File > New Connection > Container.

Figura 4- Selecionando a stack Ruby on Rails.



Fonte: acervo pessoal.

3- Nomeie, como desejar, o seu container, preenchendo o campo Name. O nome escolhido aqui é "POS", o qual significa Programação Orientada a Serviços. Abaixo desse campo, há um conjunto de stacks, que podem ser instaladas em seu container.

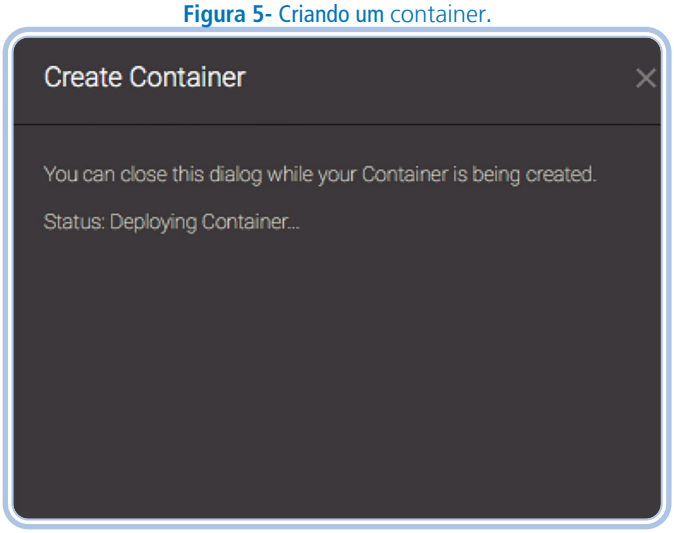
4- Para facilitar a localização da stack que desejamos, preencha o campo de busca com a palavra "ruby" (sem aspas). Serão apresentadas apenas as stacks que possuem a linguagem Ruby.

5- Selecione a stack com os seguintes dizeres: "Ruby Development Stack with RVM and Ruby on Rails preinstalled, Ubuntu 16.04"; e, por fim, clique no botão Create.

6- Em seguida, o Codeanywhere irá criar um container com o Ubuntu Linux 16.04, Ruby 2.1.2, NodeJS 0.10.25, Rails 4.1.6 e o Ruby Version Manager (RVM) 1.25.30.

7- Enquanto a janela apresentada na Figura 5 for apresentada, aguarde. Quando a criação do container for concluída, a tela ilustrada na Figura 6 será apresentada.

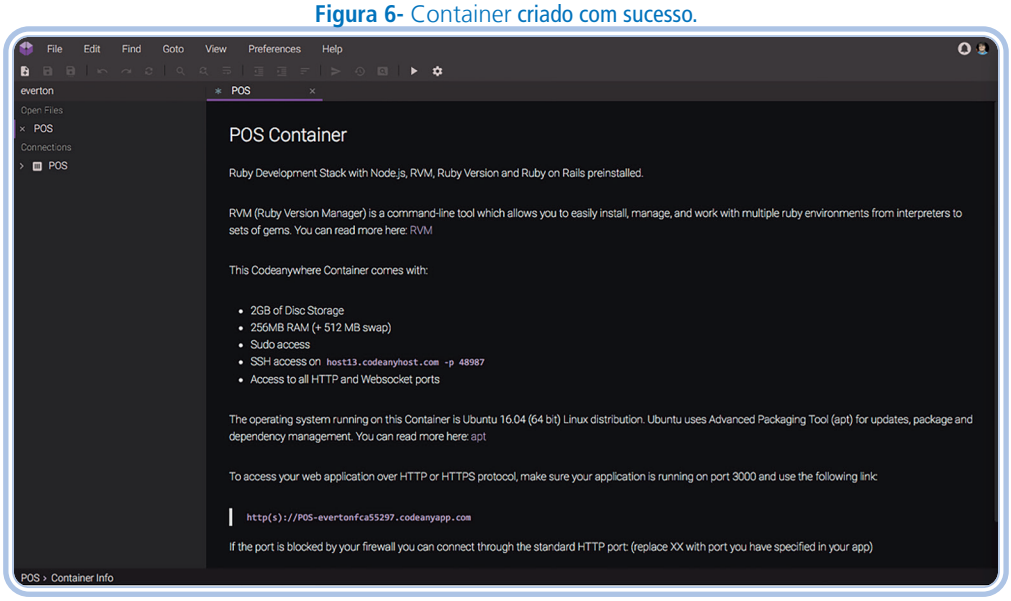
Figura 5- Criando um container.



Fonte: acervo pessoal.

8- Em seguida, o Codeanywhere irá criar um container com o Ubuntu Linux 16.04, Ruby 2.5.1, NodeJS 8.12.0, Rails 5.2.1 e o Ruby Version Manager (RVM) 1.29.4. Enquanto a janela apresentada na Figura 5 for apresentada, aguarde. Quando a criação do container for concluída, a tela ilustrada na Figura 6 será apresentada.

Figura 6- Container criado com sucesso.



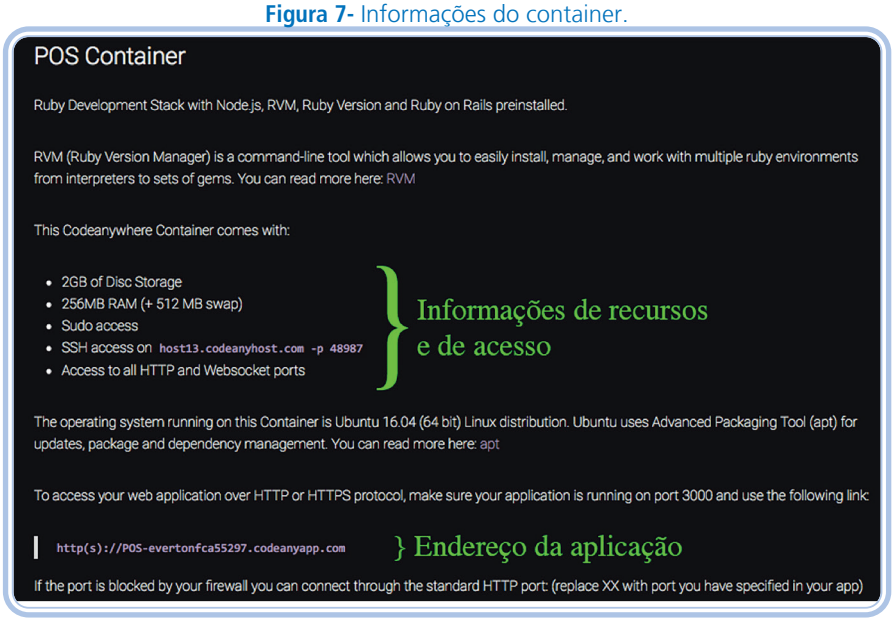
Fonte: acervo pessoal.

ATIVIDADE

2) Siga as orientações da aula para criar um container no Codeanywhere.

A tela exibida a seguir, ilustrada na Figura 7, fornece algumas informações úteis sobre o seu container.

Figura 7- Informações do container.



Fonte: Disponível em http://POS-evertonfca55297.codeanyapp.com.

Das informações exibidas na Figura 7, podemos destacar que nosso container possui a seguinte configuração: 2 GB de armazenamento (HD), 256 MB de memória RAM com mais 512 MB de SWAP. Além disso, nessa tela também existe o endereço, no qual podemos acessar nossa aplicação, quando ela estiver em execução.

RESUMINDO

Esta Aula 1 apresentou uma visão geral da plataforma de desenvolvimento on-line (PDO) Codeanywhere. A partir dela, aprendemos como criar uma conta gratuitamente e um container, a fim de desenvolver nossos projetos. Na Aula 2, continuaremos este conteúdo, explorando os recursos da PDO. Até mais!

Aula 2 - Conhecendo o ambiente de desenvolvimento - Parte 2

Olá, caro(a) estudante!

Nesta Aula 2, iremos aprender mais sobre o Codeanywhere, uma plataforma de desenvolvimento on-line (PDO) que nos permite construir nossos projetos sem a necessidade de instalação e de configuração de softwares. Na Aula I, criamos uma conta gratuitamente e, também, um container usando uma stack. Agora, vamos conhecer as funções do Codeanywhere e saber como compartilhar um container.

Objetivos de aprendizagem

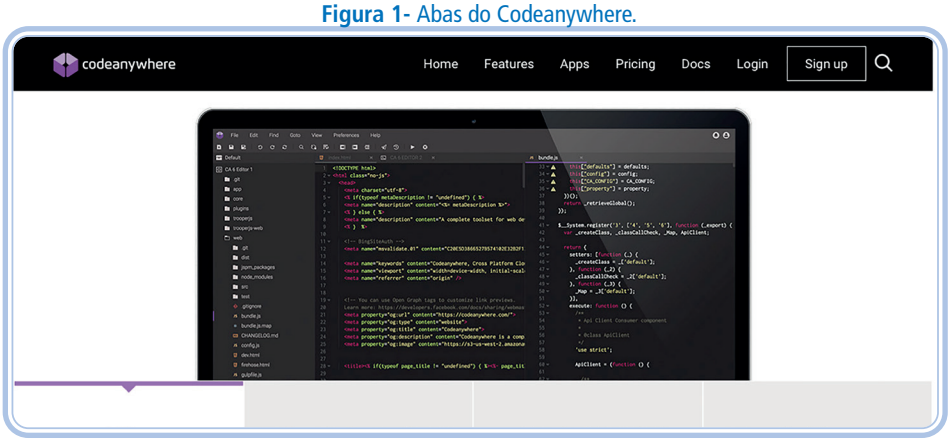
- Conhecer as funções e os recursos do Codeanywhere.
- Conhecer alguns comandos do terminal.
- Compartilhar um container com outras pessoas.
- Conhecer o dashboard do Codeanywhere.

Desenvolvendo o conteúdo

2.1 Organização da janela

O Codeanywhere organiza as suas janelas em abas, assim como fazem a maioria dos navegadores de Internet atualmente. É possível alternar as abas, ao clicar em uma, e, no caso de clicar no "x" de uma aba específica, você fechará essa aba, conforme observamos na Figura 1. Nessa figura, há uma linha de cor roxa marcando a aba ativa, ou seja, aquela que você está visualizando no momento.

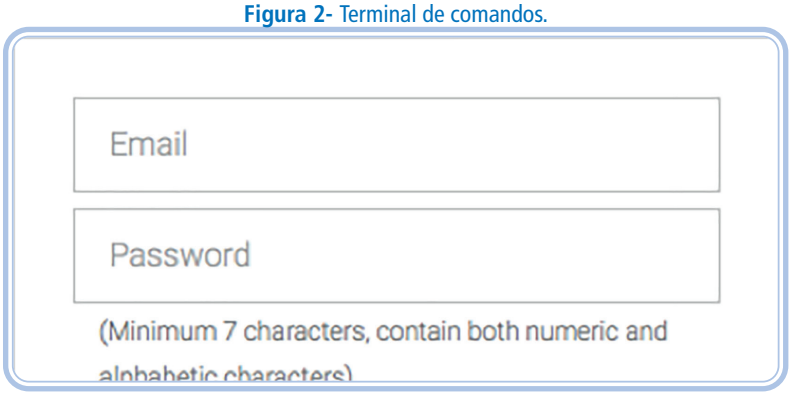
Figura 1- Abas do Codeanywhere.



Fonte: acervo pessoal.

A aba da esquerda, também chamada de POS, é o terminal de comandos do nosso container (ilustrado na Figura 2). Nele, podemos executar comandos Linux, comandos do framework Rails, acessar banco de dados, IRB, Rails Console, além de muitos outros softwares e funcionalidades.

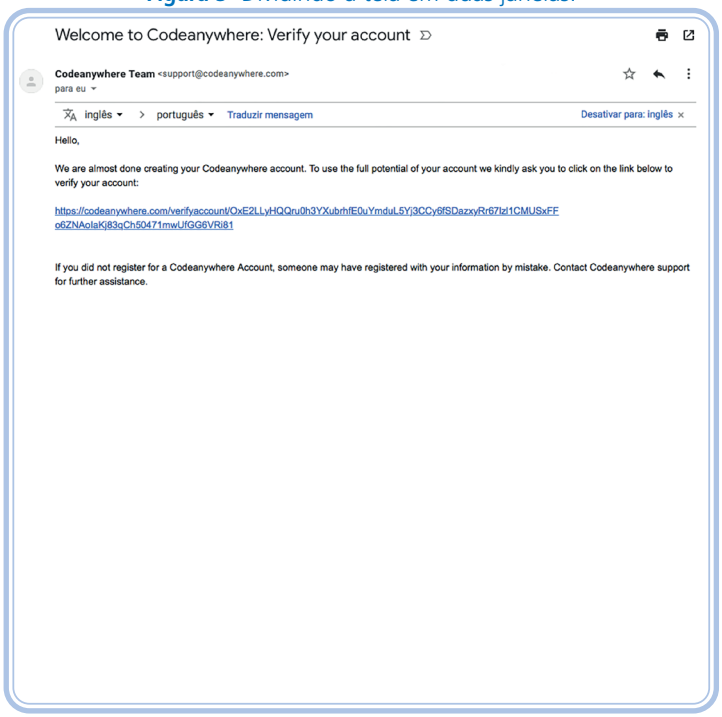
Figura 2- Terminal de comandos.



Fonte: acervo pessoal.

Dada a grande importância do terminal de comandos, sugerimos que você o mantenha sempre visível. Para isso, configure o editor do Codeanywhere para exibir mais de uma janela ao mesmo tempo. Para realizar essa configuração, clique no item View, que aparece no menu superior. Em seguida, selecione a opção Layout e, por fim, clique em Rows: 2, como podemos ver na Figura 3.

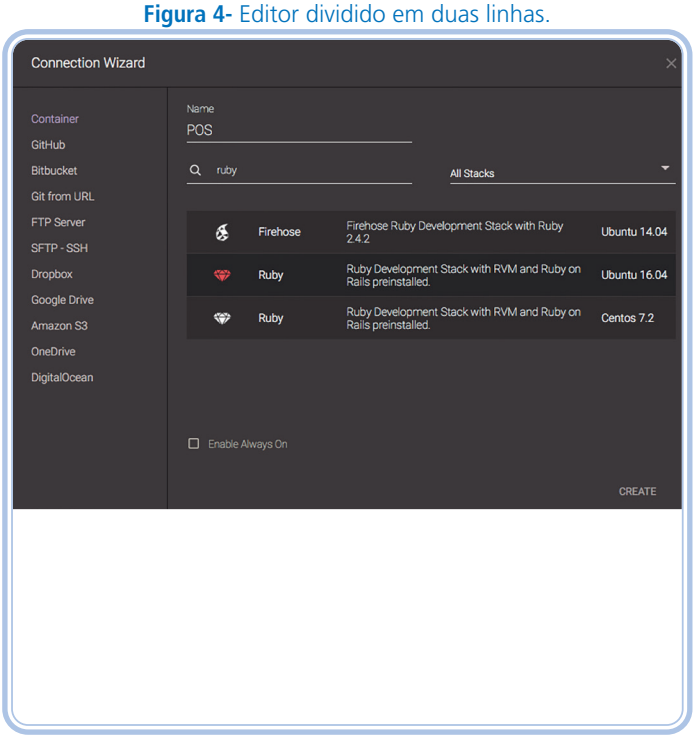
Figura 3- Dividindo a tela em duas janelas.



Fonte: acervo pessoal.

Feita essa ação, o editor do Codeanywhere deve ser exibido, como na Figura 4. Perceba que ele foi dividido em duas linhas, e cada uma delas permite abrir um conjunto de abas.

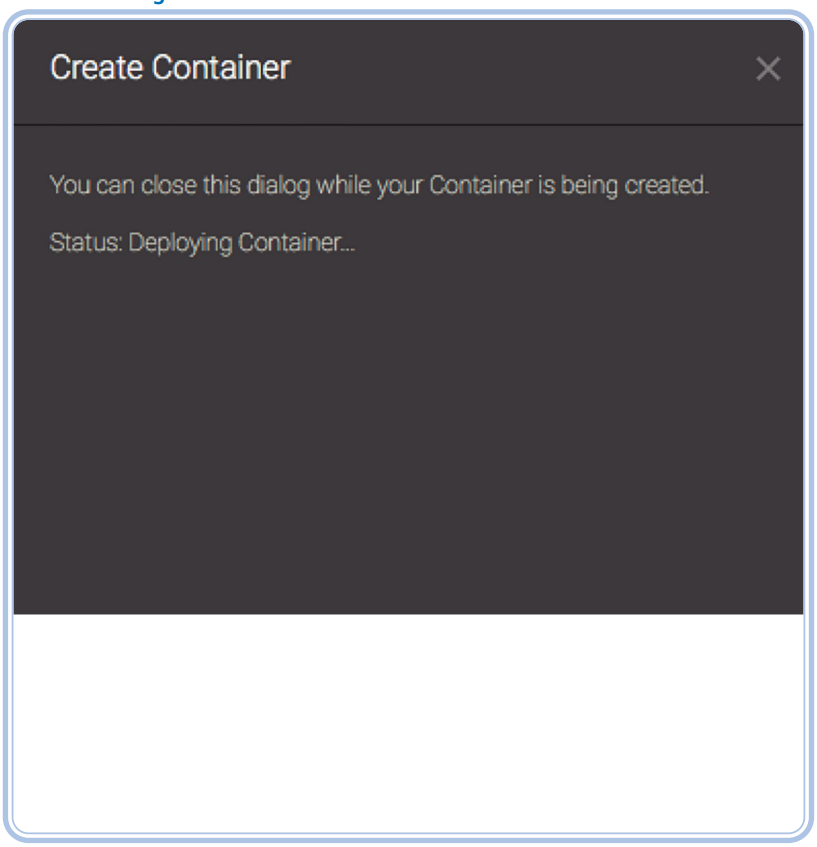
Figura 4- Editor dividido em duas linhas.



Fonte: acervo pessoal.

Em seguida, clique e segure a primeira aba POS (do terminal de comandos) e a arraste para a segunda linha. Feito isso, o editor do Codeanywhere deve ser exibido, como na Figura 5.

Figura 5- Editor exibindo duas abas simultaneamente.



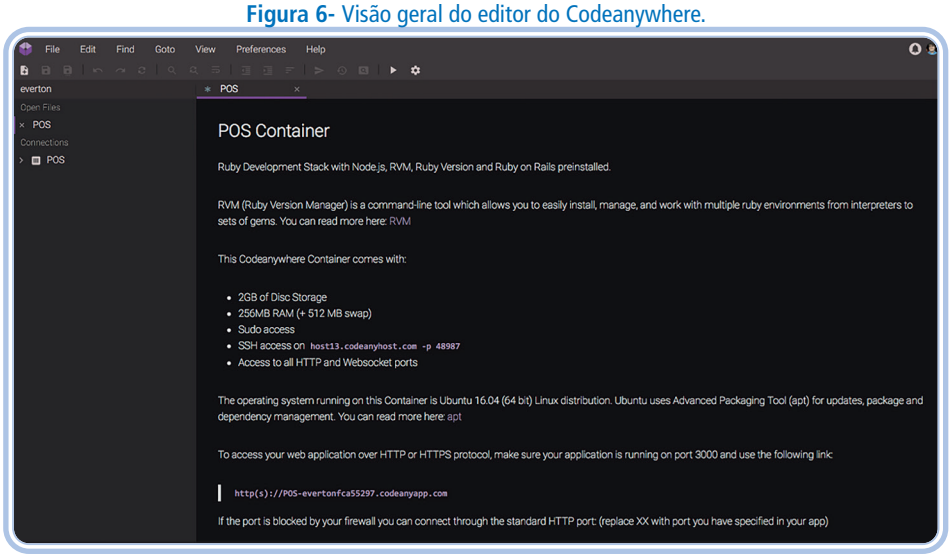
Fonte: acervo pessoal.

Vale salientar que você sempre pode fechar as abas que desejar, abrir outras abas, movê-las entre as duas linhas ou até mesmo usar outras disposições de janelas disponíveis através do menu Views > Layout.

2.2 Conhecendo o Codeanywhere

O editor do Codeanywhere, ilustrado na Figura 6, pode ser dividido em quatro partes: menu superior, gerenciador de arquivos, editor de código e terminal de comandos. A seguir, descrevemos essas partes.

Figura 6- Visão geral do editor do Codeanywhere.



Fonte: acervo pessoal.

O menu superior é similar ao encontrado em outros softwares, fornecendo opções típicas de gerenciamento de arquivos, edição, visualização, preferências e ajuda. No canto direito do menu superior, há dois ícones: o primeiro permite visualizar as notificações, enquanto o segundo exibe algumas informações sobre sua conta, além de permitir ir para o dashboard (vide Seção 6).

O gerenciador de arquivos tem funções típicas de manipulação de arquivos e diretórios, como a navegação entre pastas, a visualização de arquivos, a criação de novos arquivos e diretórios, a exclusão, o download, a clonagem, a ação renomear, entre outras funções.

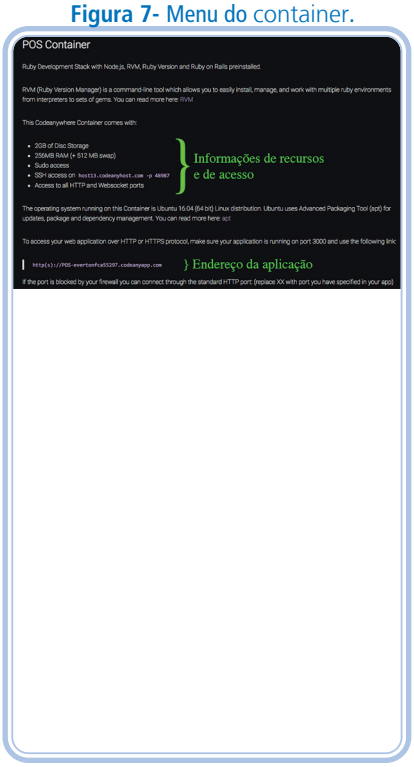
O editor de código é a janela na qual podemos manipular o conteúdo do arquivo que está sendo apresentado. Ele possui coloração de código, numeração de linhas, indentação de código, entre outras funções.

Nessa perspectiva, de acordo com o que abordamos anteriormente, o terminal de comandos é onde podemos interagir com o sistema operacional e executar comandos relativos ao Ruby on Rails.

No gerenciador de arquivos, há um item que representa o seu container (logo abaixo de Connections). Ao clicar com o botão direito do mouse sobre esse item, o menu ilustrado na Figura 7 será aberto. Esse menu conta com algumas opções importantes, das quais podemos destacar:

- SSH Terminal: abre a janela do terminal de comandos.
- Turn On/Off: liga/desliga o container (lembre-se que o container é como um computador virtual, portanto, ele pode ser ligado e desligado).
- Restart: reinicia o container.
- Info: exibe as informações do container.
- Rename: permite alterar o nome do container.
- Refresh: atualiza arquivos e diretórios do container.
- Create file: cria um arquivo.
- Create folder: cria um diretório.
- Download: permite fazer o download do container.
- Upload: permite fazer upload de arquivos para o container.
- Destroy: apaga o container.

Figura 7- Menu do container.



Fonte: acervo pessoal.

2.3 Conhecendo o terminal de comandos

Estudamos que o terminal de comandos é uma ferramenta muito importante para o desenvolvimento de projetos de software. Contudo, muitas pessoas não sabem como usar o terminal por serem usuários do sistema operacional Windows. Dessa forma, nesta seção, vamos fazer breves apontamentos sobre o terminal de comandos.

Agora, você deve estar visualizando esta linha no seu terminal de comandos:

```
cabox@box-codeanywhere:~/workspace$
```

A principal informação fornecida por essa linha é **~/workspace**, indicando que nós estamos no diretório **workspace**, o qual, por sua vez, está dentro do diretório home do nosso usuário (~). Há outras informações disponíveis, contudo, não terão relevância nesta nossa disciplina: **cabox** é o usuário que estamos usando e **box-codeanywhere** é o nome de host da nossa máquina virtual. O terminal de comandos possui uma infinidade de comandos disponíveis, mas iremos nos dedicar apenas aos que serão úteis para esta nossa disciplina.

Para ver os arquivos e os diretórios do diretório no qual estamos atualmente, devemos usar o comando **ls**. Ao executá-lo, você deverá observar a seguinte resposta:

```
cabox@box-codeanywhere:~/workspace$ ls
xpto.
```

Desse modo, temos um diretório chamado **xpto** dentro do diretório **workspace**, que é a pasta onde nos encontramos atualmente. Para entrar nesse diretório, devemos usar o comando **cd**, seguido do nome do diretório:

```
cabox@box-codeanywhere:~/workspace$ cd xpto
cabox@box-codeanywhere:~/workspace/xpto$
```

Agora, nosso diretório atual é **xpto**, que está dentro do diretório **workspace**, o qual está dentro da pasta home do nosso usuário (**~**). Esse é o diretório onde se encontra o nosso projeto Rails, criado pelo comando **rails new xpto**.

Caso você, por algum motivo, não consiga ver ou não saiba qual é o diretório atual, use o comando **pwd** apresentado aqui:

```
cabox@box-codeanywhere:~/workspace/xpto$ pwd
/home/cabox/workspace/xpto
```

Observe que a resposta do comando **pwd** é o caminho completo de onde estamos atualmente; estamos operando no diretório **xpto**, que está dentro de **workspace**, que está dentro de **cabox**, que está dentro de **home**.

No momento, estamos no diretório **xpto**, o qual está dentro da pasta **workspace**. Portanto, podemos dizer que **workspace** é o diretório pai do diretório **xpto**. O comando também pode ser usado para voltarmos para o diretório pai, ou seja, para o diretório. Para isso, use o comando **cd..**, conforme apresentado abaixo.

```
cabox@box-codeanywhere:~/workspace/xpto$ cd ..
cabox@box-codeanywhere:~/workspace$
```

Note que, usando o comando **cd..** , voltamos para o diretório **workspace**.

Muitos dos comandos que iremos usar durante nossas aulas devem ser executados dentro da pasta do projeto.

ATIVIDADE

1) Siga as orientações acima parar criar um projeto Rails e execute todos os comandos citados.

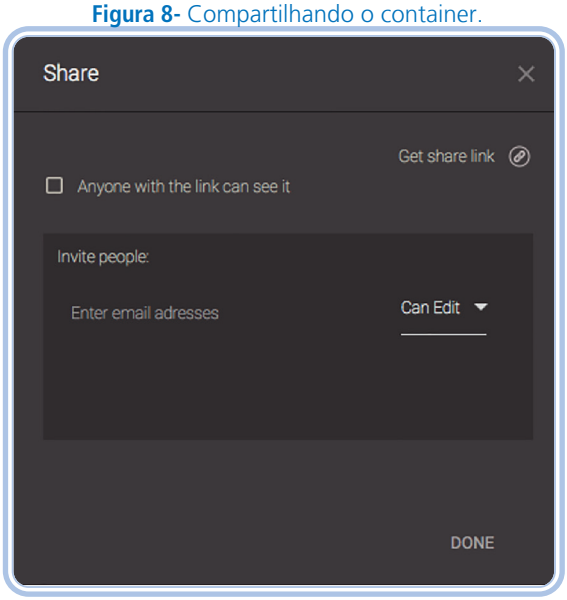
2.4 Compartilhando o *container*

Uma função muito importante do Codeanywhere é a possibilidade do compartilhamento de containers, ou seja, é permitir que outras pessoas acessem o seu container. Isso pode ser muito útil quando você precisar da ajuda de alguém para superar algum problema em seu projeto. Basta compartilhar o container com o seu professor ou um amigo, e, dessa forma, ele terá mais facilidade em te auxiliar.

Para compartilhar seu container, basta clicar na opção Share, que aparece no

menu do container na Figura 7. Após isso, o formulário ilustrado na Figura 8 será apresentado.

Figura 8- Compartilhando o container.



Fonte: acervo pessoal.

Há duas formas de compartilhar seu container: (i) através de um link; ou (ii) através de convite.

- Para compartilhar através de um link, habilite a opção Anyone with the link can see it (qualquer um com o link pode ver) e clique em Get share link. Depois disso, um link será gerado e você deverá enviá-lo para as pessoas com as quais você quer compartilhar seu container.

- A segunda forma de compartilhamento é através de convite. Para tanto, na seção Invite people (Figura 9), preencha o campo Enter email address com o endereço de e-mail da pessoa com a qual você quer compartilhar o container. No campo ao lado, selecione se essa pessoa poderá editar (Can Edit) ou apenas visualizar (Can View) seu container.

ATIVIDADE

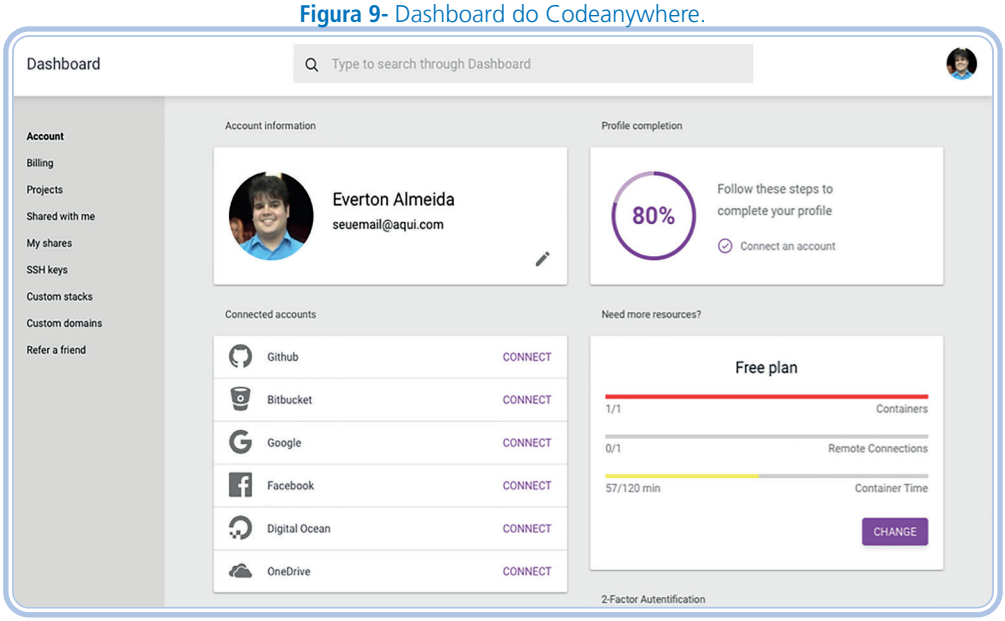
2) Siga as orientações da aula e compartilhe, através de um link, seu container com o professor.

2.5 Dashboard

O dashboard (https://codeanywhere.com/dashboard) é um painel que nos permite ter acesso a várias funções de configuração do Codeanywhere. Na página inicial do dashboard, ilustrada na Figura 9, temos uma visão geral da conta, informações como nome, e-mail e foto do usuário, como as contas que podem ser conectadas e os serviços e detalhes sobre o plano utilizado.

Estamos usando uma conta gratuita, portanto, não precisamos fornecer dados de cobrança para o Codeanywhere.

Figura 9- Dashboard do Codeanywhere.



Fonte: Disponível em https://codeanywhere.com/dashboard

De todas as opções disponíveis no menu do lado esquerdo do dashboard, podemos destacar algumas úteis para as nossas aulas:

- Account: permite alterar configurações da conta como nome, nome de usuário, endereço de e-mail, senha, apagar a sua conta, entre outras funções;

- Projects: permite que você crie, abra, edite ou apague projetos (containers);

- Shared with me: permite visualizar os containers compartilhados com você;

- My shares: permite visualizar os containers que você está compartilhando com outras pessoas.

RESUMINDO

Nesta Aula 2, encerramos o tema Codeanywhere, o qual foi iniciado na aula passada. Aqui, foram descritas algumas funções e recursos dessa plataforma, a maneira de usar o terminal de comandos e de compartilhar containers e conhecer o seu dashboard. Na próxima aula, estudaremos o Protocolo HTTP. Até lá!

Aula 3 - Protocolo HTTP

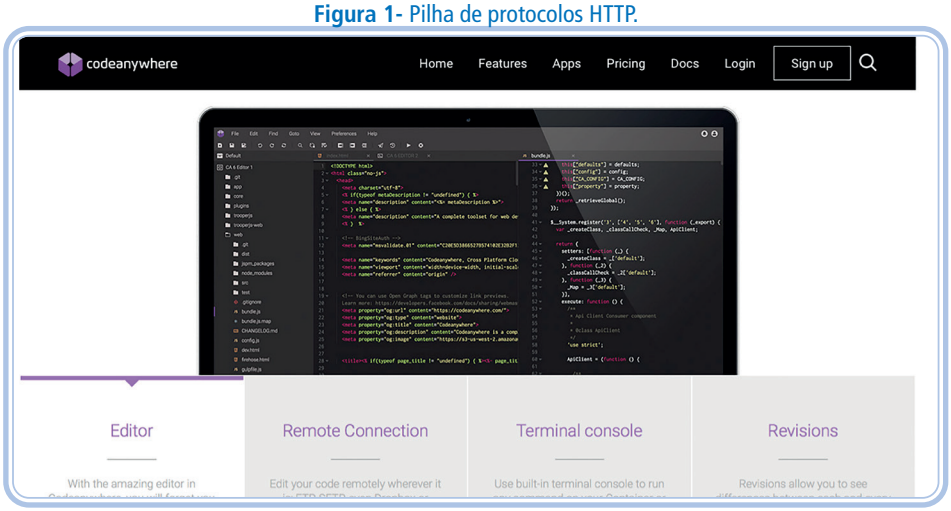
Objetivos de aprendizagem

- Conhecer o protocolo HTTP.
- Compreender o funcionamento e a composição desse protocolo.
- Identificar a estrutura da requisição e da resposta HTTP.

Desenvolvendo o conteúdo

O protocolo HTTP, sigla para Hypertext Transfer Protocol, que em português significa Protocolo de Transferência de Hipertexto, é utilizado diariamente para navegar na World Wide Web.

Figura 1- Pilha de protocolos HTTP.

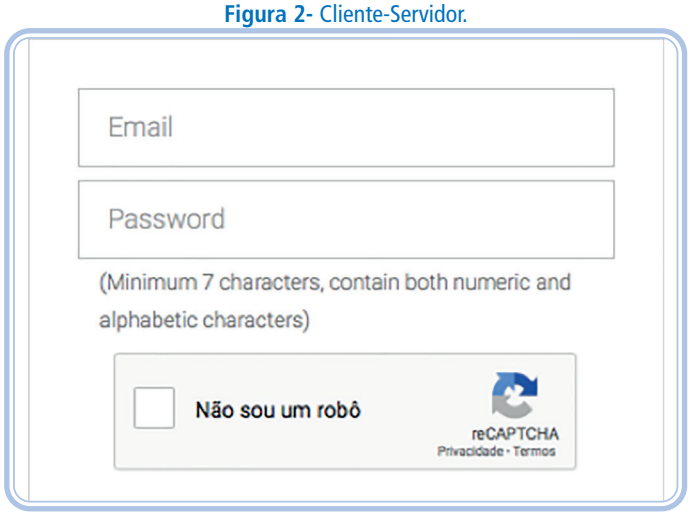


Fonte: acervo pessoal.

Conforme ilustrado na Figura 1, o protocolo HTTP roda sobre o protocolo TCP, sigla para Transmission Control Protocol, ou Protocolo de Controle de Transmissão, que é um protocolo de transmissão de dados confiável, com checagem de erros, no qual os pacotes são entregues de forma ordenada. Para garantir essa confiabilidade, tal protocolo acaba se tornando um pouco mais lento do que o protocolo UDP, sigla para User Datagram Protocol, o qual não tem essa confiabilidade, mas é mais veloz na entrega dos pacotes.

O protocolo TCP, por sua vez, roda sobre o protocolo IP, sigla para Internet Protocol, ou Protocolo de Internet, que é o mais importante protocolo de comunicação na Internet atualmente. Por fim, todos esses protocolos rodam sobre a Ethernet, a qual consiste em uma família de tecnologias de redes de computadores locais, metropolitanas e de longo alcance.

Figura 2- Cliente-Servidor.



Fonte: acervo pessoal.

O protocolo HTTP segue os modelos Cliente/Servidor e Requisição e Resposta. No modelo Cliente/Servidor, demonstrado na Figura 2, temos o cliente a solicitar serviços, e o servidor a oferecer serviços aos clientes.

Figura 3- Requisição e Resposta.

Fonte: acervo pessoal.

No modelo Requisição e Resposta, o cliente envia uma requisição para o servidor, solicitando algum recurso ou serviço, e o servidor responde ao cliente com uma resposta. A Figura 3 ilustra as partes que compõem a requisição e a resposta.

Ambas são compostas por cabeçalho (headers) e corpo (body). É importante que você memorize as traduções dessas duas palavras, pois, em breve, utilizaremos programas em inglês, que se referem a cabeçalho e a corpo como headers e body.

Portanto, no protocolo HTTP, temos os clientes, que solicitam serviços ao servidor através de requisições, estas compostas por headers e body. E os servidores respondem tais requisições dos clientes, através de respostas também compostas por headers e body.

3.1 O protocolo HTTP

O protocolo HTTP é baseado em requisições e respostas realizadas entre clientes e servidores. A seguir, apresentamos as requisições e as respostas desse protocolo.

3.2 Requisição HTTP

Nesta seção, vamos conhecer mais detalhes sobre a requisição do protocolo HTTP. Ao abrir abrir o seu navegador e acessar o site do IFRN, através do endereço http://www.ifrn.edu.br, o seu navegador irá enviar uma requisição HTTP para o servidor do IFRN, como se vê na Figura 4.

Figura 4- Requisição HTTP.

Fonte: acervo pessoal.

Como sabemos, essa requisição possui cabeçalho e corpo. O cabeçalho da requisição é semelhante ao mostrado na Figura 4. Esse é um exemplo de cabeçalho de uma requisição HTTP, que, entre algumas informações disponíveis, duas são fundamentais para a nossa disciplina: **o método HTTP** e o **recurso solicitado**.

O recurso é aquilo que o cliente está solicitando. Por exemplo, quando acessamos o portal do IFRN, o recurso solicitado é a página principal do portal, que será apresentada em nosso navegador.

O método HTTP indica o que deve ser feito com o recurso. Quando, simplesmente, acessamos um endereço na Internet, digitando-o no navegador, estamos enviando uma requisição com o método HTTP GET, que acessa um recurso no servidor. Numa requisição HTTP GET, o corpo da requisição não possui informação alguma, pois o cabeçalho já contém todos os dados necessários para que a requisição seja atendida pelo servidor.

Geralmente, ao preenchermos e enviarmos um formulário na Internet, estamos enviando uma requisição HTTP POST, que é usada para criar um novo recurso. Nesse caso, o corpo da requisição contém os dados do novo recurso a ser criado no servidor.

Além do GET e do POST, os mais utilizados, temos outros três métodos: o DELETE, usado para apagar um recurso; e o PUT ou PATCH, para atualizar um recurso, e, nesses casos, o corpo da requisição contém os novos dados do recurso a ser atualizado. No Quadro 1, você verifica os principais métodos HTTP.

Quadro 1 - Métodos HTTP

Método	Corpo	Descrição
GET	Vazio	Acessa/recupera um recurso.
POST	Dados do novo recurso	Cria um novo recurso.
DELETE	Vazio	Apaga um recurso.
PUT/PATCH	Novos dados do recurso	Atualiza um recurso.

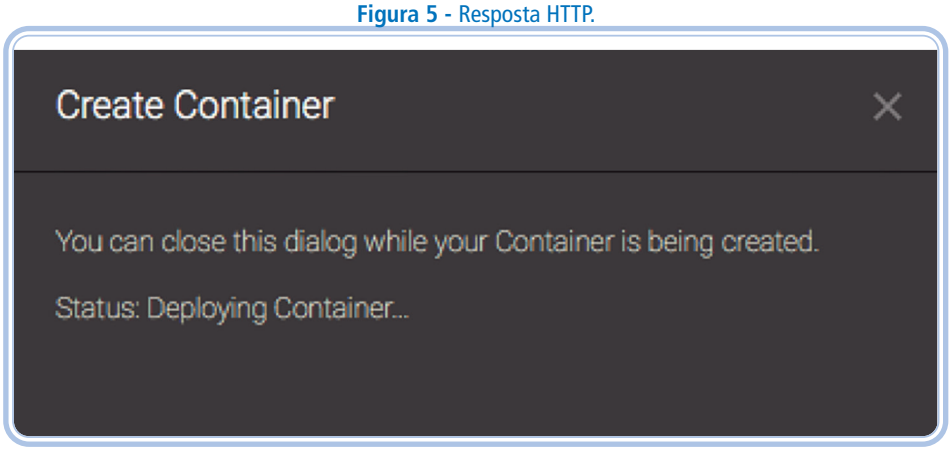
Fonte: Adaptado, disponível em https://nandovieira.com.br/entendendo-um-pouco-mais-sobre-o-protocolo-http

Ainda existem alguns outros métodos HTTP, como HEAD, CONNECT, OPTIONS e TRACE. Por enquanto, você só deve memorizar a utilidade de cada um dos métodos HTTP estudados nesta aula, pois eles serão amplamente usados em aulas futuras.

3.3 Resposta HTTP

A fim de detalhar a resposta HTTP, destacamos, dentre as informações disponíveis no cabeçalho dela, o HTTP Status Code, ou Código de Resposta HTTP.

Figura 5 - Resposta HTTP.



Fonte: acervo pessoal.

O código de resposta HTTP indica a situação no atendimento da requisição enviada pelo cliente. Ou seja, se a requisição foi atendida com sucesso ou se houve algum problema. O Quadro 2 apresenta alguns códigos de resposta HTTP.

Quadro 2 - Códigos de Resposta HTTP

Código	Nome	Descrição
200	OK	Sucesso
301	Moved Permanently	O recurso foi permanentemente movido para outra URI.
302	Found	O recurso foi temporariamente movido para outra URI.
304	Not Modified	O recurso não foi alterado.

401	Unauthorized	Não autorizado.
403	Forbidden	Proibido.
404	Not Found	Não localizado.
405	Method Not Allowed	Método não permitido.
410	Gone	Indisponível.
500	Internal Server Error	Erro no servidor.

Fonte: Adaptado pelo Autor. Disponível em https://nandovieira.com.br/entendendo-um-pouco-mais-sobre-o-protocolo-http

Caso a requisição tenha sido atendida com sucesso, será enviada uma resposta HTTP com o código 200. Caso o usuário não tenha permissão para acessar o recurso solicitado, será enviada uma resposta HTTP com o código 401. Quando um cliente solicita um recurso e ele não é localizado pelo servidor, é enviada uma resposta HTTP com o código 404, o famoso erro 404, que você já deve ter visto. Caso haja um erro interno no servidor para atender a uma requisição, o servidor irá enviar uma resposta HTTP com o código 500.

Você pode estar se perguntando: o que contém no corpo de uma resposta HTTP? O corpo de uma resposta HTTP pode variar muito de conteúdo, mas, geralmente, quando fazemos uma requisição GET a um endereço na Internet, o corpo da resposta contém aquilo que será apresentado no navegador do usuário.

Leituras complementares

- Como funciona uma requisição HTTP?, disponível em: https://www.youtube.com/watch?v=fhAXgcD21iE

- Resumo, disponível em: https://www.youtube.com/watch?v=YX9bMbtK8Lk

Atividade de aprendizagem

1) Liste os quatro principais métodos HTTP, definindo-os.

2) Descreva, com suas palavras, o significado dos códigos de resposta HTTP 200, 304 e 404.

Aula 4 - O processo administrativo e seus elementos

Objetivos

Ao final desta aula, você será capaz de:

- Comparar os formatos de XML e JSON.
- Apresentar o formato JSON.
- Criar objetos JSON a partir de exemplos.

Desenvolvendo o conteúdo

Nesta Aula 4, iremos apresentar o JSON e o XML, formatos legíveis baseados em texto com suporte para criação, leitura e decodificação em aplicativos do mundo real. Ambos são notação de texto, hierárquica e independente de linguagem de programação para troca de dados.

Apesar dos traços comuns, eles diferem em muitos aspectos, como tipos de dados, verbosidade, pilha de ferramentas etc. Enquanto XML é uma linguagem de marcação baseada em texto e especializada em transações business to business na World Wide Web, o JSON é um padrão aberto leve, formato para troca de dados que é estendido do JavaScript.

XML e JSON

XML, sigla para eXtensive Markup Language, ou Linguagem de Marcação Extensiva, é um formato de dados baseado em texto derivado de SGML (ISO 8879), escrito de forma semelhante, seguido por HTML. O XML existe

há anos e foi desenvolvido, principalmente, para superar os desafios da publicação eletrônica em grande escala.

Ele se responsabiliza por terceirizar dados e armazená-los em formato de texto simples, em vez de integrá-los no documento HTML, o que o torna ideal para representar dados hierárquicos, como documentos, transações, faturas, livros e muito mais. Consiste em um formato de intercâmbio de dados independente, que codifica documentos em um formato que seja legível por computadores e por humanos. Ou seja, trata-se de maneira flexível de criar formatos de informações e de compartilhar dados estruturados na World Wide Web.

A principal vantagem do XML é ser independente de plataforma. Assim, os usuários podem obter dados de outros programas, como o SQL, convertê-los em XML e compartilhá-los com outras plataformas. Em síntese, é uma tecnologia orientada a documentos, a qual fornece a capacidade de armazenar e de exibir dados em formato legível por máquina e por humanos. É uma metalinguagem sem semântica inerente, o que a torna um formato ideal para criar dados ad-hoc e documentar formatos de informações.

Nesse contexto, o **JSON**, sigla para JavaScript Object Notation, ou Notação de Objetos JavaScript, é um formato de troca de dados baseado em texto, o qual usa tipos de dados de texto e números para representar objetos. É um formato de padrão aberto, baseado no subconjunto da linguagem de programação JavaScript, mas pode ser usado independente de linguagem de programação.

Consiste na maneira de transmitir objetos de dados, que consistem em tipos de dados de matriz e de pares de valor de atributo entre clientes e servidores na web. Ele usa um formato legível para representar estruturas de dados simples no código baseado em aplicativo da web.

Devido à sua flexibilidade, o JSON é mais adequado para intercâmbio de dados entre aplicativos da Web e serviços da web. Como uma linguagem de marcação, o XML só adiciona informações extras a um texto simples, enquanto o JSON, como o nome sugere, é representa objetos de dados.

Ele também é usado em ambientes de programação de desktop e de servidor. Ao contrário do XML, o JSON adota uma abordagem simples para representar dados de estrutura, sem notação matemática e algoritmos complexos, além

de ser fácil de aprender e, portanto, de criar mais páginas interativas.

Em outros termos, o problema de um é a vantagem do outro. A sintaxe XML é livre de semântica, mas é detalhada e complexa, podendo dificultar o uso de todos os aplicativos. Ele foi projetado para melhorar a legibilidade, não para ser eficiente. A sintaxe JSON é muito mais compacta com sua semântica estabelecida, tornando-o um formato de dados preferencial sobre XML; por isso, adotaremos o JSON, bem como nos determos a esse padrão na próxima seção.

Leituras Complementares

- XML e JSON: O que é, semelhanças, diferenças e utilização. Disponível em: https://www.youtube.com/watch?v=hBnhsr4Eyl8:

JSON

Como mencionamos anteriormente, o JSON é um padrão de formatação de dados em formato texto para troca de informações. Atualmente, tem sido amplamente adotado e utilizado, por, facilmente, ser lido e escrito por humanos e compreendido e gerado por software.

Existem duas estruturas básicas no JSON: o conjunto de pares chave/valor e a lista. No par chave/valor, ilustrado abaixo, a chave é aquilo que aparece antes dos dois pontos, e o valor é o que aparece depois.

A chave é sempre delimitada por aspas duplas, o valor nem sempre. Adiante, você vai conferir mais detalhes a esse respeito. O padrão JSON também permite que você tenha um conjunto de pares chave/valor. Nesse caso, os pares são separados por vírgula. Os pares chave/valor são sempre delimitados por chaves.

Veja o JSON a seguir:

```
{
  “nome”: “Everton Almeida”,
  “altura”: 1.75
}
```

No exemplo acima, um objeto que tem a chave "nome" tem como valor "Everton Almeida", e a chave "altura" tem valor 1.75.

Além dos pares chave/valor, existem as listas, que se assemelham muito aos arrays conhecidos em linguagens de programação e correspondem a um conjunto de valores separados por vírgula e delimitados por colchetes, como exemplificado a seguir.

```
[1, 1, 2, 3, 5, 8, 13]
```

Objetos em JSON

Para representar um objeto em JSON, basta usar um conjunto de pares chave/valor, sendo um par para cada propriedade do objeto. Por exemplo, o objeto da classe Produto em Ruby, com os atributos nome "Café especial", preço 3.65 e marca "Café do Vale Dourado", seria escrito da seguinte maneira:

```
produto = Produto.new(nome: "Café especial", preco: 3.65,
marca: "Café do Vale Dourado")
```

Esse objeto, em JSON, seria representado assim:

```
{
  "nome": "Café especial",
  "preco": 3.65,
  "marca": "Café do Vale Dourado"
}
```

Primeiramente, qualquer representação de objeto em JSON é delimitada por chaves, e, dentro delas, colocamos os pares chave/valor equivalendo a cada uma das propriedades do objeto. Observe que as chaves e os valores são separados por dois pontos, as chaves são delimitadas por aspas duplas e os pares são separados por vírgula. Essa é a representação de um objeto em JSON.

Você pode estar se perguntando se esse código em JSON poderia ser escrito em uma única linha. A resposta é sim, essa representação com indentação é usada apenas para facilitar a nossa leitura.

Nesse aspecto, uma lista de valores em JSON é representada, separando os valores por vírgula e delimitando toda a lista com colchetes. Os valores da lista não precisam ser numéricos, podem ser strings e até mesmo uma lista de objetos JSON, a qual citamos anteriormente.

Tipos de valores em JSON[1]

Os valores em JSON podem ser de sete tipos diferentes: string, number, object, array, true, false e null. Caracterizamos esses tipos a seguir.

A string é um conjunto de caracteres delimitados por aspas duplas.

Exemplo 1:

```
{ “string”: “json” }
```

Os valores numéricos são números sem delimitadores.

Exemplo 2:

```
{ “numero”: 10 }
```

Os objetos são conjuntos de pares chave/valor delimitados por chaves.

Exemplo 3:

```
{ “nome”: “Pelé”, “camisa”: 10 }
```

Os arrays são listas de valores delimitados por colchetes.

Exemplo 4:

```
{ “fibonacci”: [1, 1, 2, 3, 5, 8, 13] }
```

1

Já os valores true e false são valores booleanos, que representam verdadeiro e falso, respectivamente. E null representa nulo.

Exemplo 5:

```
{ “verdadeiro”: true, “falso”: false, “nulo”: null }
```

Strings são sempre delimitadas por aspas duplas (exemplo 1). Dentro das aspas, podemos representar quaisquer caracteres e números, inclusive espaços em branco (exemplo 2). Para representar aspas duplas numa string em JSON, é preciso usar a contra barra antes das aspas, para indicar que estas não são o encerramento da string (exemplo 3). Além disso, outros caracteres de escape podem ser usados, como quebra de linha (exemplo 4), por exemplo:

```
{
  “exemplo 1”: “json”,
  “exemplo 2”: “Ruby on Rails”,
  “exemplo 3”: “Diga \”Hello!\””,
  “exemplo 4”: “Olá\n”
}
```

Valores numéricos são representados sem delimitadores. Dentre os valores numéricos, podem ser representados números inteiros positivos, negativos e, também, números reais, nos quais ao invés da vírgula devemos usar o ponto.

```
{
  “positivo”: 33,
  “negativo”: -33,
  “real”: 3.923
}
```

Exemplos em JSON

Na Tabela 1, apresentamos exemplos de representação de dados na forma tabela e a sua equivalência em JSON

Tabela 1 - Dados do exemplo 1

Nome	E-mail	Idade	Altura	Situação
Antônio Silva	antonio@mail.com	42	1.75	false
Maria do Carmo	maria@mail.com	31.	1.62	true

Fonte: Autoria própria (2018).

Vamos começar representando a primeira linha da tabela em JSON. Iniciamos delimitando o objeto com chaves, e, em seguida, usamos os cabeçalhos de cada coluna como chaves delimitadas com aspas duplas. Sucessivamente, colocamos os valores de cada coluna no local correspondente, delimitando as strings com aspas duplas também. Por fim, separamos todos os pares com vírgulas.

Isso se repetirá para a segunda linha; novamente, iniciamos delimitando o objeto com chaves, e, dentro delas, usamos as colunas da tabela para formar os pares chave/valor, delimitando as chaves e as strings com aspas duplas. Por fim, basta separar os pares com vírgula. Porém, isso não é tudo. Você consegue imaginar o que falta fazer?

Na Tabela 1, temos duas linhas de dados e representamos cada uma delas como objetos JSON, contudo, esses códigos estão separados e precisamos uni-los em um único código. Para isso, basta usar a notação de listas em JSON, colocando os objetos entre colchetes e os separando com vírgula. O objeto JSON equivalente à Tabela 1 seria:

```
[
  {
    “nome”: “Antônio Silva”,
    “email”: “antonio@mail.com”,
    “idade”: 42,
    “altura”: 1.75,
    “situacao”: false
  },
  {
    “nome”: “Maria do Carmo”,
    “email”: “maria@mail.com”,
    “idade”: 31,
    “altura”: 1.62,
    “situacao”: true
  }
]
```

Agora, vamos representar os dados do exemplo 2, na Tabela 2 (abaixo), em JSON. A diferença é a lista de valores na coluna "Setores".

Tabela 2 - Dados do exemplo 2

Nome	E-mail	Setores
Antônio Silva	antonio@mail.com	RH, Financeiro
Maria do Carmo	maria@mail.com	Gestão, TI

Fonte: Autoria própria (2018).

Aqui, temos a representação das duas linhas da tabela em JSON. Observe que, no valor da chave "setores", estamos usando uma lista de strings. Para representar toda a tabela do exemplo 2, basta colocarmos os objetos JSON numa lista, delimitando-os com colchetes e separando os objetos com vírgula. O JSON que representa a Tabela 2 é:

```
[
  {
    "nome": "Antônio Silva",
    "email": "antonio@mail.com",
    "setores": ["RH", "Financeiro"]
  },
  {
    "nome": "Maria do Carmo",
    "email": "maria@mail.com",
    "setores": ["Gestão", "TI"]
  }
]
```

Em nosso exemplo 3, temos a representação dos dados das tabela 3 e 4:

Tabela 3 - Funcionários do exemplo 3

Nome	E-mail	Setores
Antônio Silva	antonio@mail.com	1,2
Maria do Carmo	maria@mail.com	3,4

Fonte: Autoria própria (2018).

Tabela 4 - Setores do exemplo 3

Setores		
ID	Nome	Sigla
1	Recursos Humanos	RG
2	Financeiro	FIN
3	Gestão	GES
4	Tecnologia da Infomação	TI

Fonte: Autoria própria (2018).

Nesse exemplo 3, cada funcionário possui um conjunto de setores, e cada setor tem um conjunto de dados: id, nome e sigla.

Para representar os setores de um funcionário, usamos uma lista e, dentro dela, colocamos cada um dos objetos correspondentes. Esse código é a representação em JSON da primeira linha da Tabela 3 (Funcionários), com os dados correspondentes da Tabela 4 (Setores).

Para finalizar, colocamos o objeto JSON, apresentado nas Tabelas 3 e 4, numa lista, incluímos uma vírgula no final para separar os dois objetos na lista, e, agora, temos a representação do segundo objeto em JSON finalizando a lista com um colchete.

```
[
  {
    "nome": "Antônio Silva",
    "email":"antonio@mail.com",
    "setores": [
      {
        "id": 1,
        "nome": "Recursos Humanos",
        "sigla": "RH"
      },
      {
        "id": 2,
```

```
      "nome": "Financeiro",
      "sigla": "FIN"
    }
   ]
  },
  {
   "nome": "Maria do Carmo",
   "email": "maria@mail.com",
   "setores": [
    {
     "id": 3,
     "nome": "Gestão",
     "sigla": "GES"
    },
    {
     "id": 4,
     "nome": "Tecnologia da Informação",
     "sigla": "TI"
    }
   ]
  }
]
```

LEITURAS COMPLEMENTARES

Padrão JSON, disponível em: https://www.youtube.com/watch?v=p_By9yedhbo.

RESUMINDO

Nesta Aula 4, abordamos os padrões de representação de dados JSON e XML. Após compará-los, aprofundamos nossos estudos no JSON. Falamos sobre as suas estruturas básicas, os tipos de valores aceitos, bem como representamos uma série de dados, através de exemplos, em JSON.

Atividade de aprendizagem

1) Baseado nos conteúdos estudados nesta aula, desenvolva um JSON que apresente os seus dados para esse semestre: nome, matrícula, e-mail, disciplinas que está cursando e nome dos docentes que ministram as respectivas disciplinas.

Aula 5 - Consumindo serviços com um cliente REST - Parte 1

Objetivos

Ao final desta aula, você será capaz de:

- Compreender o conceito de REST.
- Utilizar um cliente REST.
- Aprender a consumir serviços de uma API usando um cliente REST.

Desenvolvendo o conteúdo

O que é REST?

Primeiramente, a World Wide Web é uma arquitetura REST. Então, se você utiliza a Internet, seja acessando o Google, Instagram, Facebook etc, quer dizer que já está experimentando a utilidade do REST.

O REST é utilizado em ambientes distribuídos orientados a serviços, onde tradicionalmente faríamos uso de tecnologias como SOAP, CORBA, DCOM, RMI ou RPC, etc. Todas essas tecnologias foram criadas como um mecanismo para permitir que os componentes de software/aplicativos se comuniquem por meio de uma rede, portanto, o trabalho que deve ser feito por esse software pode ser distribuído em vários computadores, em uma rede local ou remota.

O REST pode ser visto como um sucessor do SOAP. Ele não é um framework estritamente definido, mas sim um princípio de design para criar aplicativos, que se comunicam entre si através de uma rede.

Todas as tecnologias supracitadas ainda estão em uso, embora apresentem deficiências significativas. Algumas são limitadas a redes locais e não funcionam na Internet, algumas não são suficientemente flexíveis, algumas conectam os componentes de software muito próximos, de modo que não podem evoluir de forma independente.

O REST usa HTTP e HTTPS como o protocolo de transporte, e não para codificar as chamadas dos serviço no corpo de uma mensagem. REST não define um formato específico das mensagens trocadas, mas tenta simplificar as coisas, restringindo todas as ações aos verbos HTTP padrão, como: GET, POST, PUT, DELETE. Também aproveita a noção de recursos - entidades que são identificadas exclusivamente por suas respectivas URI.

A ideia geral do REST é que os componentes de software podem usar a Web da mesma maneira que os humanos com seu navegador. Interagir com hiperlinks e documentos e trocar dados de um jeito simples, não precisando de informações extras sobre como esses dados trocam de trabalhos.

Um recurso em um aplicativo REST é uma entidade definida por um URI, ativada para executar um ou mais dos métodos HTTP padrão. Isso facilita a disponibilização de interfaces para interação direta com os seres humanos, por exemplo: fornecendo uma representação HTML; e, ao mesmo tempo, para outras aplicações, no formato JSON e/ou XML.

Portanto, o REST é útil para criar aplicativos de comunicação remotos simples, fáceis de usar, eficientes e de fácil manutenção.

.Utilizando um cliente REST

Para usar os serviços, precisamos de um cliente REST para fazer as requisições. Nesse caso, vamos utilizar o Advanced REST Client, que é uma extensão para o navegador Google Chrome.

O Advanced REST Client está disponível, através do endereço eletrônico https://install.advancedrestclient.com/install, conforme a Figura 1.

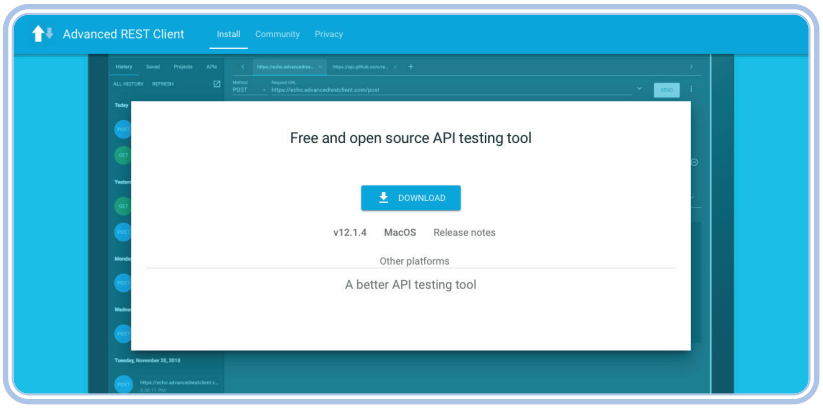


Figura 1: Página do Advanced REST Client

Para instalá-lo, abra o navegador Google Chrome e digite a seguinte url: https://chrome.google.com/webstore/detail/advanced-rest-client/hgmloofddffdnphfgcellkdfbfbjeloo (ou ainda pesquise Advanced REST Client no buscador Google). Abrirá a página da Figura 2. Clique no botão "Usar no Chrome".

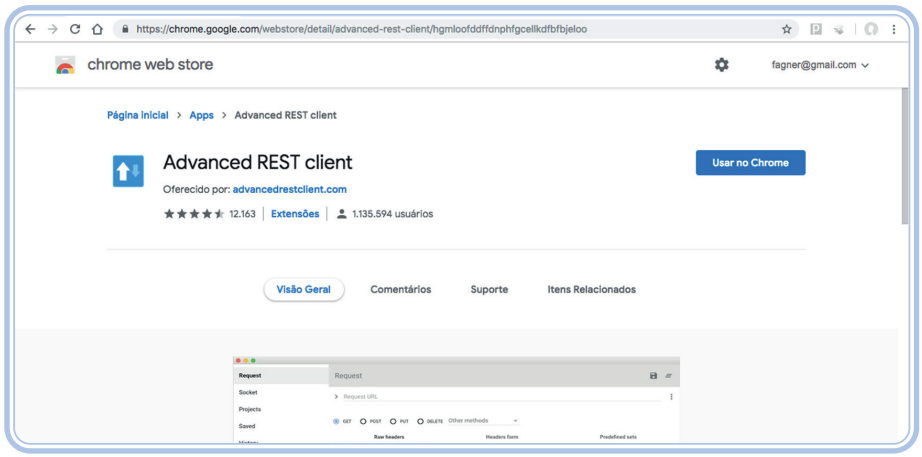


Figura 2: Página da instalação no navegador Google Chrome

Após isso, clique em "Adicionar aplicativo", conforme a Figura 3

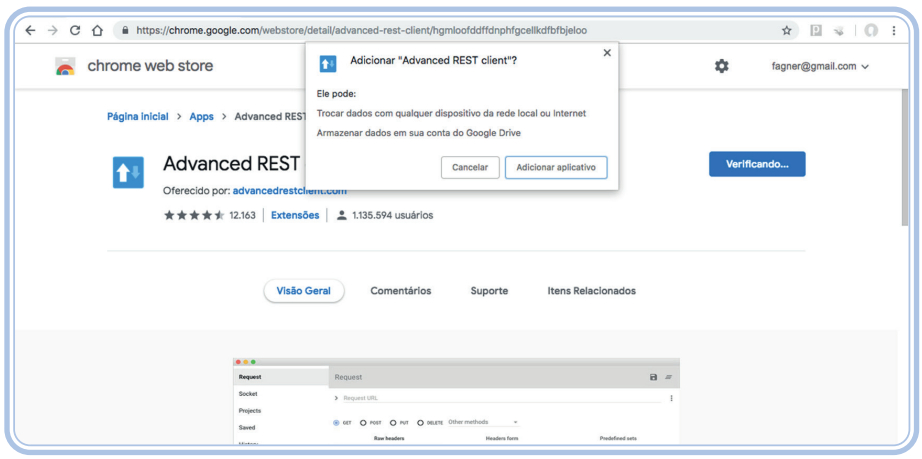


Figura 3: Processo de instalação do Advanced Rest Client

Concluída a instalação, o Advanced REST Client, ou ARC, estará disponível no navegador Google Chrome, através da página chrome://apps/, que pode ser acessada clicando no botão Apps, conforme a Figura 4.

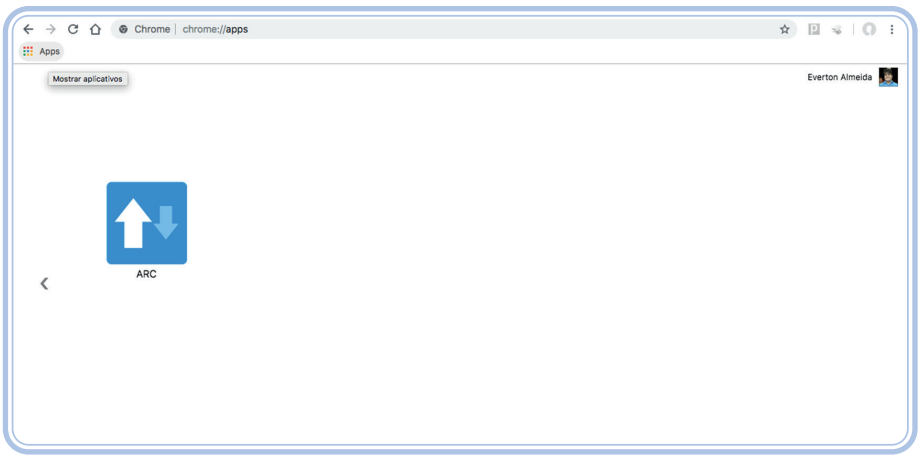


Figura 4: ARC instalado no navegador Google Chrome

Clique no botão do ARC para iniciá-lo. Caso deseje, você pode ver um pequeno tutorial; caso não, basta clicar em SKIP.

Pronto, agora, vamos testar o ARC enviando uma requisição GET para o Portal do Instituto Federal do Rio Grande do Norte (IFRN). No campo Method, mantenha a opção GET. No campo Request URL, digite o endereço http://portal.ifrn.edu.br/. Para enviar a requisição, clique em SEND.

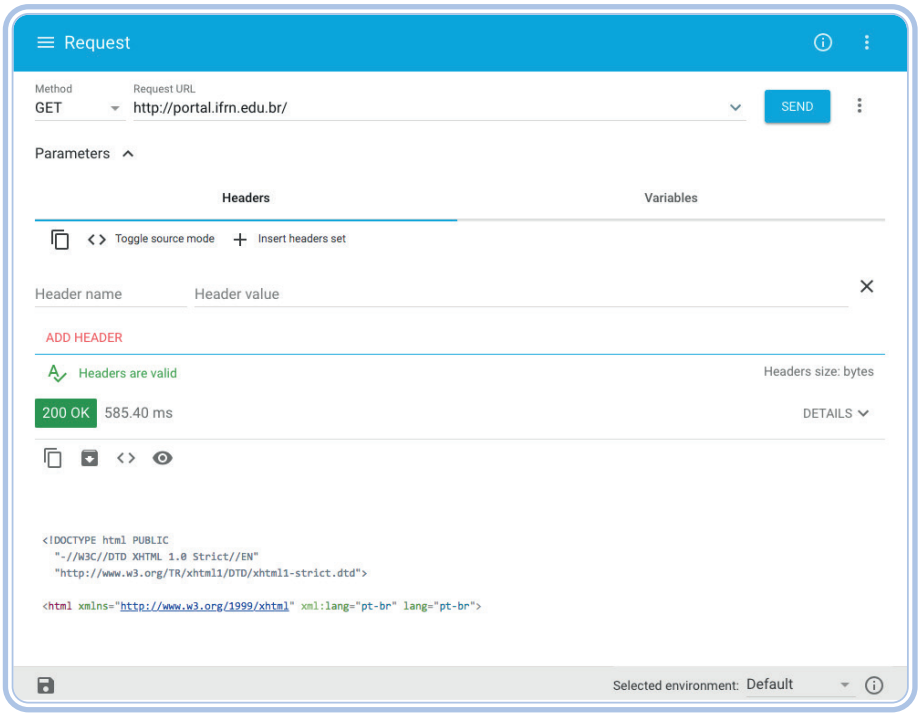


Figura 5: Resposta à requisição HTTP portal do IFRN pelo ARC

Como vemos, na Figura 5, o código da resposta HTTP: 200 significa que a nossa requisição foi atendida com sucesso. Além disso, observe o corpo da resposta, logo abaixo, que é a página inicial do Portal do IFRN.

Agora, vamos usar uma API para consultar os preços médios de veículos no mercado brasileiro. API é o acrônimo para Application Programming Interface, que é um intermediário de software, o qual permite dois aplicativos conversarem entre si. Sempre que você usa um aplicativo como o Facebook, envia uma mensagem instantânea ou verifica o clima no seu smartphone, está usando uma API. O site https://deividfortuna.github.io/fipe/ explica os detalhes sobre como consultar os serviços disponíveis nessa API.

[LEMBRE-SE!

Antes de começar a usar uma API, é importante ler a documentação dela para compreender como funciona.

Para consultarmos o preço de um veículo específico, precisamos fornecer a marca do veículo, o modelo e o ano. A API possui um serviço que lista as marcas de veículos disponíveis. Para acessar esse serviço, copie e cole, no ARC, o endereço https://fipe.parallelum.com.br/api/v1/carros/marcas. Mantenha o método GET e clique em SEND.

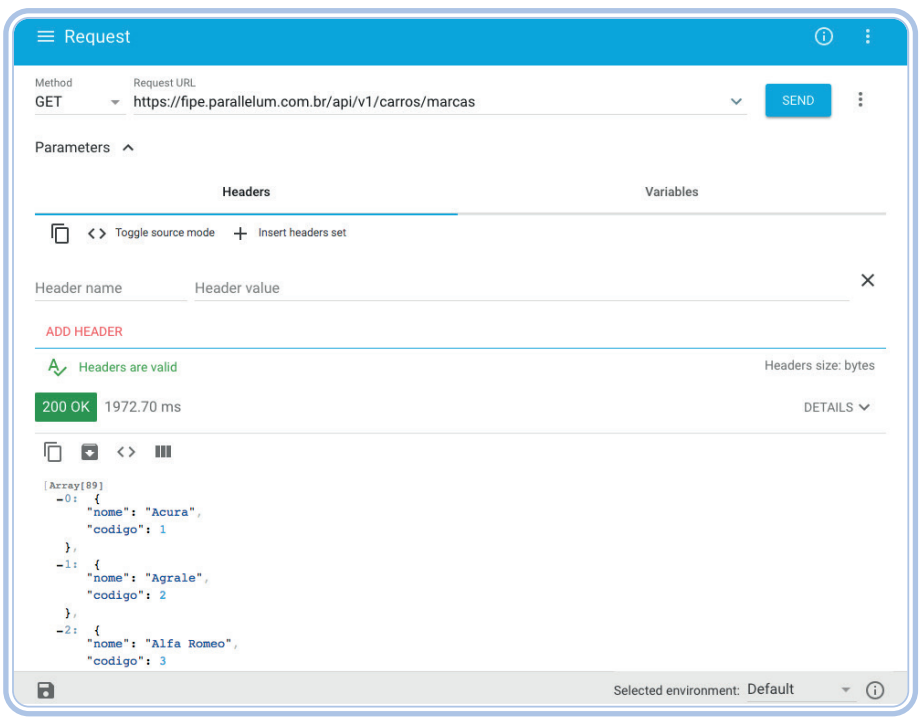


Figura 6: Resposta JSON

Na Figura 6, obtivemos a resposta em uma lista de objetos em JSON, a qual contém nomes e códigos de marcas. Vamos escolher uma marca "Alfa Romeo", por exemplo, e usar o código dessa marca para consultar os modelos de carros disponíveis para essa marca.

Vamos voltar para o site da API e observar que, para consultar os modelos de uma marca, basta adicionarmos /, seguido do código da marca e de /modelos no final do endereço. Depois, volte para o ARC e altere o endereço adicionando /3, que é o código da marca Alfa Romeo, seguido de /modelos, para consultar os modelos dessa marca. Feito isso, clique em SEND.

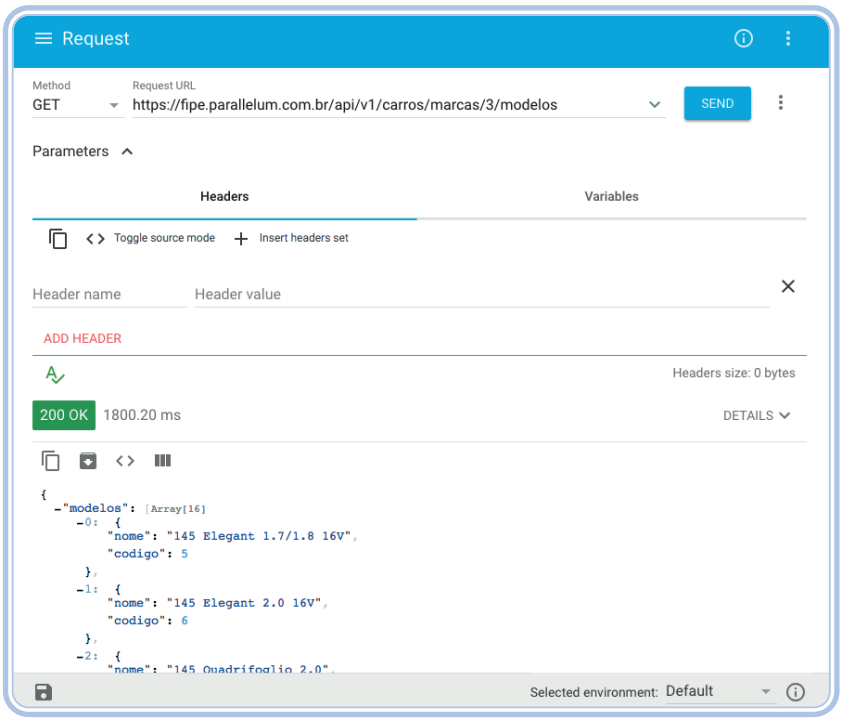


Figura 7: Resultado de veículos da marca Alfa Romeo

O resultado (Figura 7) é uma lista de objetos em JSON, que contém os nomes e os códigos de modelos de veículos da Alfa Romeo. Vamos escolher o modelo o "145 Elegant 1.7/1.8 16V", por exemplo, e usar o código dele no serviço que consulta os anos disponíveis para esse veículo.

Vamos voltar para o site da API, note que, para consultar os anos disponíveis para um modelo de carro, basta adicionarmos /, o código do modelo e, por fim, /anos. Vamos voltar para o ARC, e, no campo de endereço, adicionar /5, que é o código do modelo o qual desejamos consultar, seguido de /anos. Feito isso, clique em SEND e veja o resultado na Figura 8.

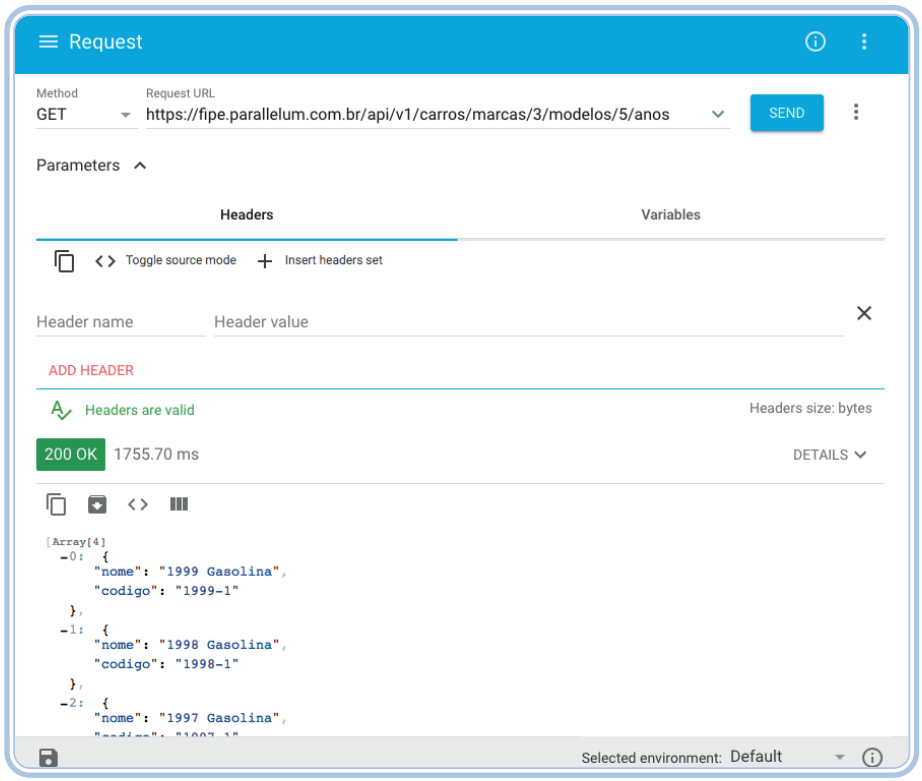


Figura 8: Resultado da consulta

Na Figura 8, o resultado é uma lista de objetos em JSON, que contém nomes e códigos. Vamos escolher o ano "1999 Gasolina", por exemplo, e usar o código do respectivo ano, no caso 1999-1, no serviço que consulta o valor do veículo.

Vamos voltar para o site da API e constatar que, para consultar o valor médio do veículo, basta adicionar ao endereço /, seguido do código do ano. Então, vamos voltar para o ARC e digitar /1999-1, como na Figura 9.

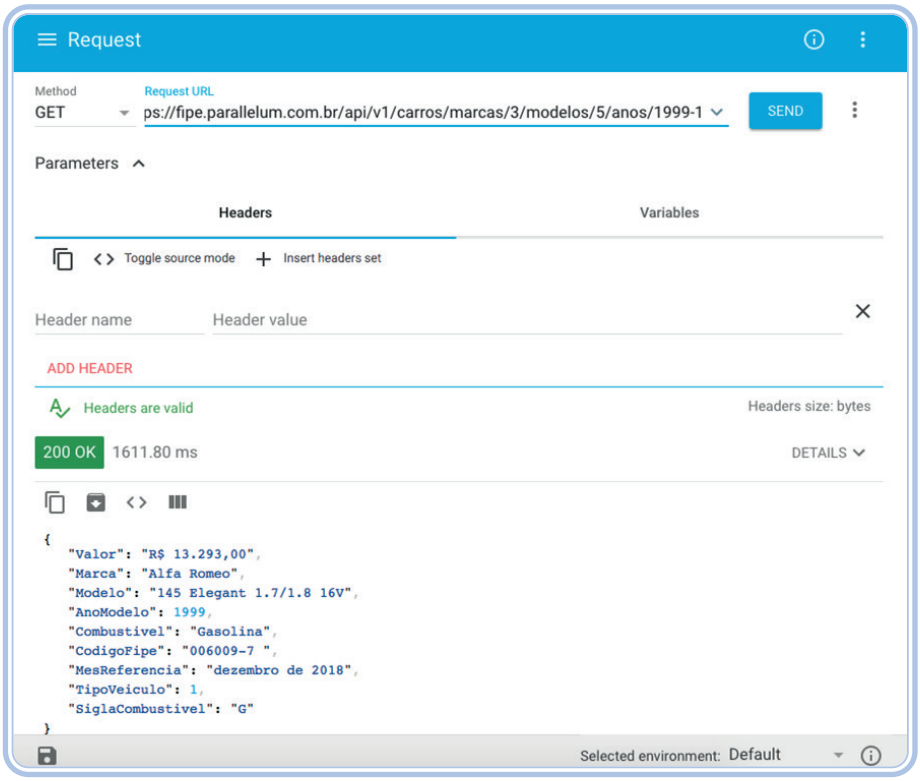


Figura 9: Resultado

Pronto! A resposta (Figura 9) é um objeto JSON com os detalhes do veículo pesquisado, que mostra, inclusive, o valor médio de mercado no Brasil, na chave Valor.

Como você deve ter percebido, para consultar os dados nessa API não precisamos fornecer nenhum tipo de credencial e nem fazer cadastros prévios. Contudo, nem sempre é assim. As APIs costumam exigir algum tipo de autenticação para que elas possam ser utilizadas. Iremos abordar esse assunto na próxima aula.

LEITURAS COMPLEMENTARES

- O que é API RESTful? Entenda aqui! Disponível em: https://blog.iset.com.br/o-que-e-api-restful-entenda-aqui/.

- Consumindo serviços com um cliente REST. Disponível em: https://youtu.be/TWfmXruzmPI.

RESUMINDO

Nesta Aula 5, conceituamos REST, o qual tem uso e manutenção simplificados e facilita a criação e a comunicação entre aplicações. Também, instalamos e utilizamos um Fliente REST, o ARC; e consumimos dados da API da tabela FIPE, através do ARC.

Na próxima aula, iremos continuar o estudo sobre as APIs. Até lá!

Atividade de aprendizagem

1) Siga as orientações estudadas e crie um objeto JSON com os detalhes de veículo da API Fipe.

Aula 6 - Consumindo serviços com um cliente REST - Parte 2

Na Aula 5, consultamos dados de uma API REST, sem a necessidade de fornecer qualquer tipo de credencial e nem de fazer cadastros prévios. Contudo, não é sempre assim, pois as APIs costumam exigir algum tipo de autenticação para serem utilizadas. Nesta Aula 6, iremos consumir serviços com um cliente REST de uma API que necessita de autenticação. Vamos lá!?

Objetivo de aprendizagem

Ao final desta aula, você será capaz de:

- Aprender a consumir serviços de uma API utilizando um cliente REST, o qual precisa de autenticação.

Desenvolvendo o conteúdo

API do SUAP

Para demonstrar uma API que precisa de autenticação, será verificada a API do SUAP, da qual você pode acessar a documentação através do endereço https://suap.ifrn.edu.br/api/docs.

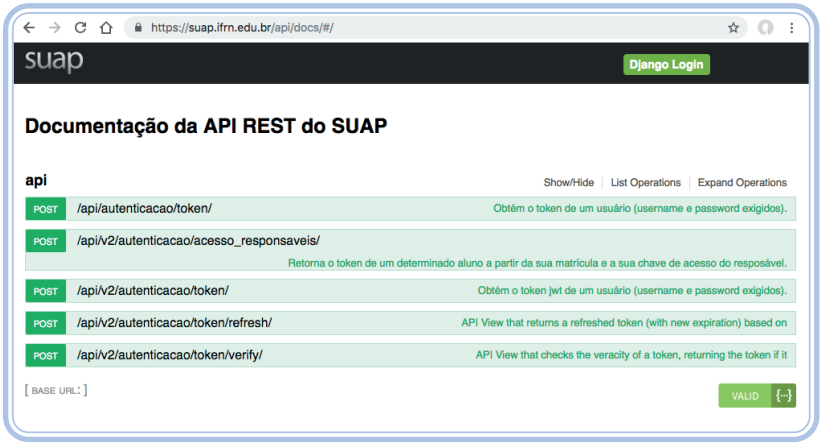


Figura 1: Página da documentação da API do SUAP

A API do SUAP exige que o usuário forneça suas credenciais do SUAP para poder acessar seus serviços. Ao fornecer suas credenciais, você receberá um token, que será usado para autorização de uso dos serviços dessa API.

Para solicitar seu token do SUAP é simples, vamos utilizar este serviço: POST /api/v2/autenticacao/token, sendo preciso fornecer nossas credenciais do SUAP para recebermos o token de acesso.

Observe que, nesse serviço, devemos enviar uma requisição POST, e, no corpo da requisição, fornecer nossas credenciais do SUAP, seguindo esse padrão em JSON.

Agora, vamos voltar para o ARC, configurar o método POST, e, no campo endereço, inserir https://suap.ifrn.edu.br/api/v2/autenticacao/token/, conforme consta na documentação. Logo, vamos adicionar uma configuração no cabeçalho, clicando em Add Header. No campo Header Name, digite Content-Type. E, no campo Header Value, digite application/json. Essa configuração define que os dados no corpo da requisição estão formatados em JSON, como percebemos na Figura 2.

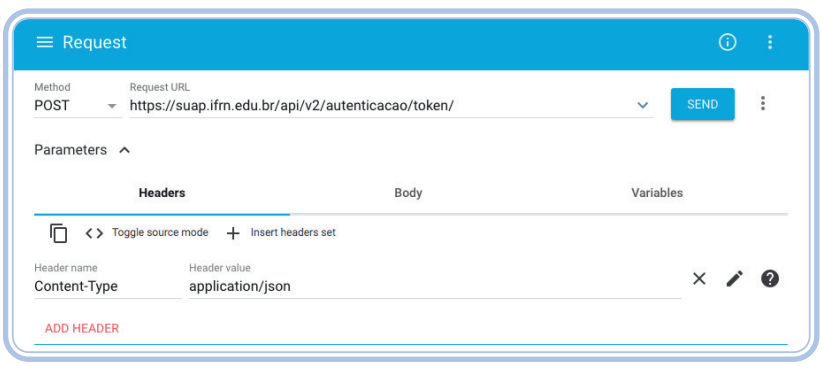


Figura 2: Configurando o Headers

Em seguida, clique em Body e forneça suas credenciais do SUAP, conforme este código em JSON:

```
{
  “username”: “Sua matrícula do SUAP”,
  “password”: “Sua senha do SUAP”
}
```

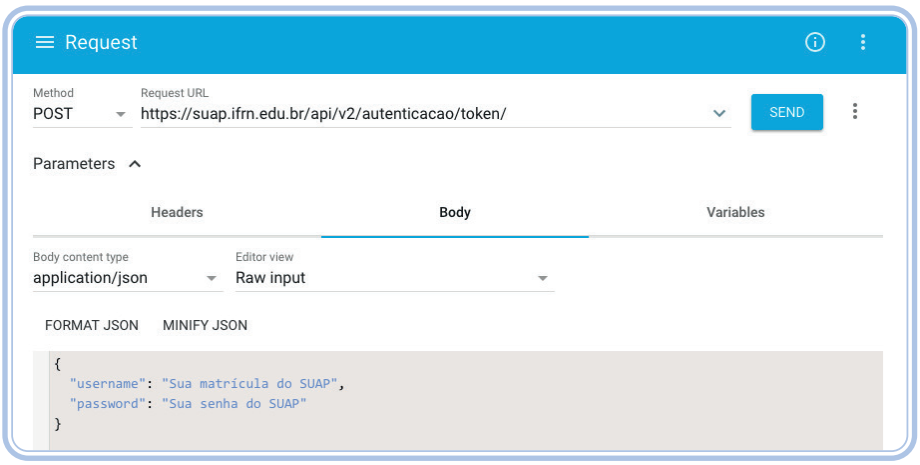


Figura 3: Adicionando credenciais no Body

Por fim, clique em SEND. Assim, será retornado o JSON com o seu token, semelhante ao resultado da Figura 4.

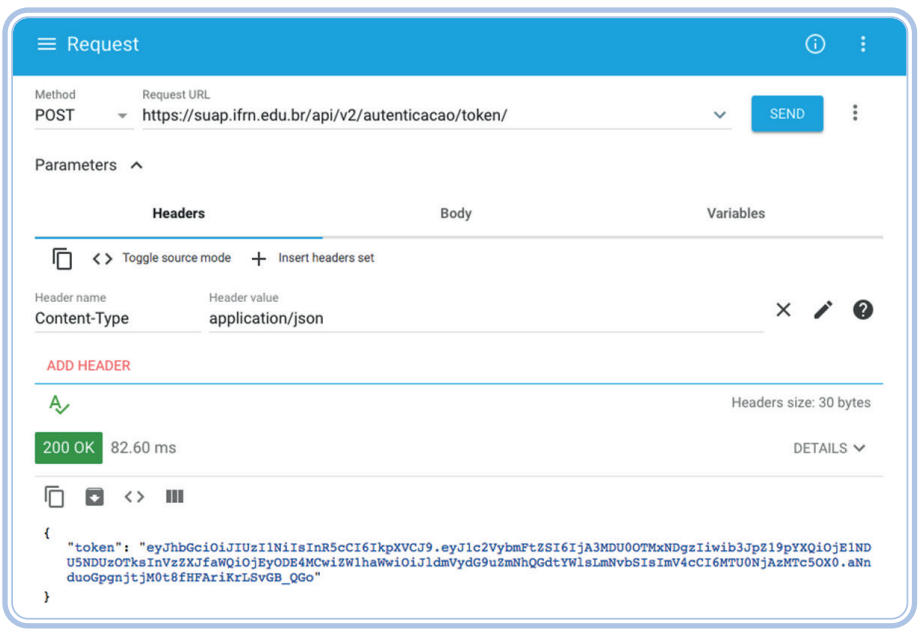


Figura 4: Retorno do token

Atividade de aprendizagem

1) Siga as orientações desta aula, a fim de obter o seu token.

De posse do seu token, volte para a documentação da API do SUAP, clique em Authorize, preencha os campos username e password com suas credenciais do SUAP e, depois, clique em Authorize, como mostra a Figura 5.

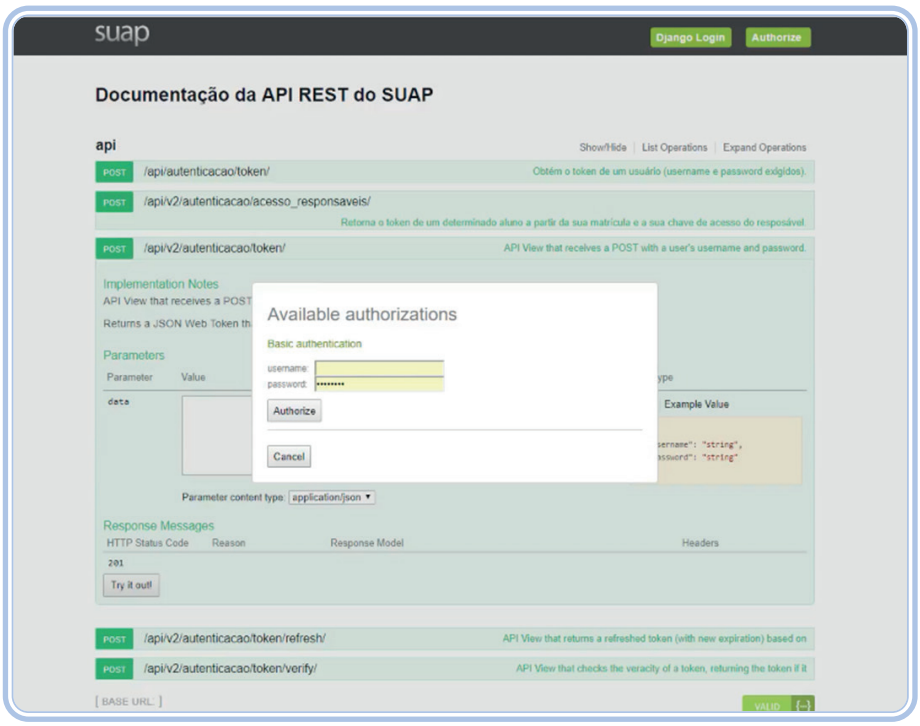


Figura 5: Autorizando acesso pelo token

Agora, você poderá ver todos os serviços que a API do SUAP fornece. Utilize o serviço GET /api/v2/minhas-informacoes/meus-dados para obter seus próprios dados no SUAP.

Volte para o ARC, altere o método para GET e use o endereço https://suap.ifrn.edu.br/api/v2/minhas-informacoes/meus-dados. Se você clicar em SEND, vai receber o erro 401 Unauthorized (Figura 6), porque não forneceu o token na requisição.

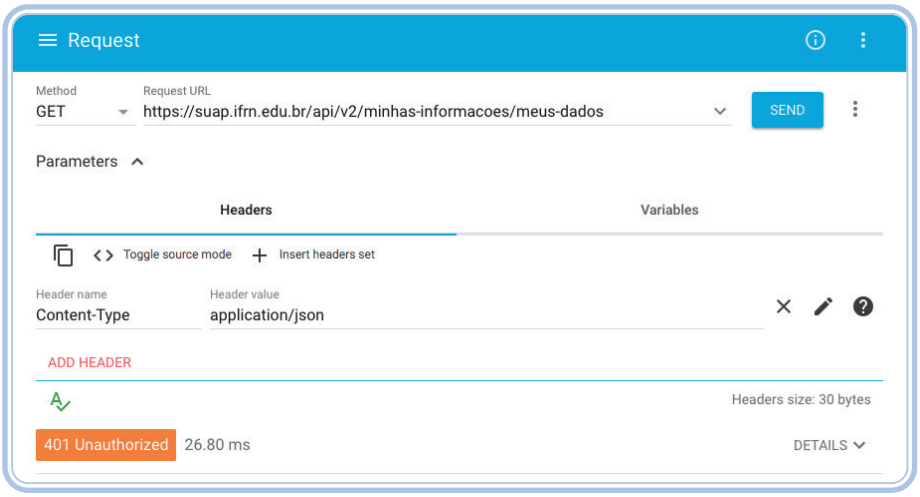


Figura 6: Requisição não autorizada

Para fornecer o token, devemos adicionar uma nova configuração ao cabeçalho, clicando em Add Header. No campo Header name, digitar Authorization; e, no campo Header value, digitar JWT seguido do token (vide Figura 7). Feito isso, clique em SEND.

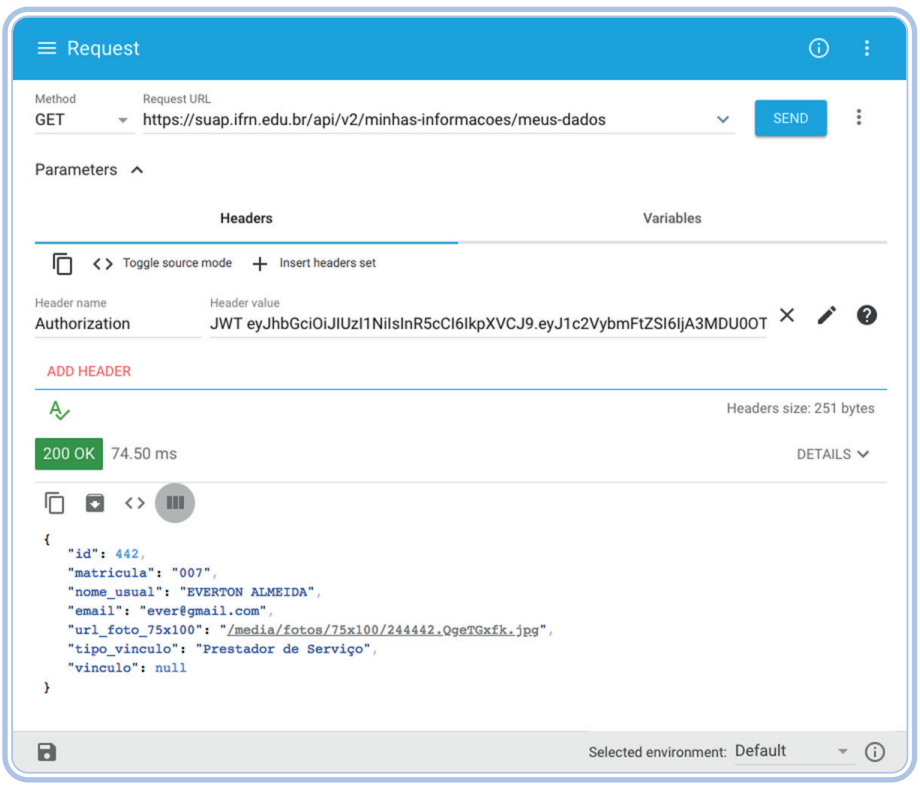


Figura 7: Objeto JSON com os dados do SUAP

Observe que a resposta é um objeto em JSON com seus dados cadastrados no SUAP. A Figura 7 apresenta um resultado, no qual temos as chaves: id, matricula, nome usual, e-mail, url_foto_75x100, tipo_vinculo e vinculo.

Leituras Complementares

- Web services RESTful: Como adicionar segurança com JWT. Disponível em: https://www.devmedia.com.br/web-services-restful-como-adicionar-seguranca-com-jwt/38990.

- Consumindo serviços com um cliente REST. Disponível em: https://youtu.be/TWfmXruzmPI.

RESUMINDO

Nesta Unidade VI, continuamos o aprendizado sobre o processo de consumir serviços de uma API usando um cliente REST, que necessita de autenticação. Para tanto, utilizamos a API do SUAP, obtendo o token de acesso e, posteriormente, os dados do usuário pelo ARC.

Atividade de aprendizagem

2) Siga as orientações estudadas nesta aula e obtenha o JSON com seus dados do SUAP.

Aula 7 - Consumindo serviços com Ruby

Objetivos de aprendizagem

- Aprender a consumir serviços utilizando a linguagem Ruby.
- Implementar scripts Ruby no Codeanywhere.

Desenvolvendo o conteúdo

Vamos iniciar esta Aula 7 acessando o nosso projeto no Codeanywhere. Para tanto, você deve ter criado o container para Ruby on Rails, o qual estudamos na Aula 1. Após criá-lo, devemos fazer o nosso projeto usando o comando no terminal.

Feito isso, instale o Bundler - o gerenciador de dependências para Ruby -, que fornece um ambiente consistente para projetos Ruby, rastreando e instalando as gems e as versões exatas necessárias. O Bundler vai nos ajudar a instalar uma gem, a qual usaremos em nosso projeto.

Para instalar o Bundler, deve-se ir ao terminal, abrir o diretório do projeto Rails e executar o seguinte comando:

```
gem install bundler
```

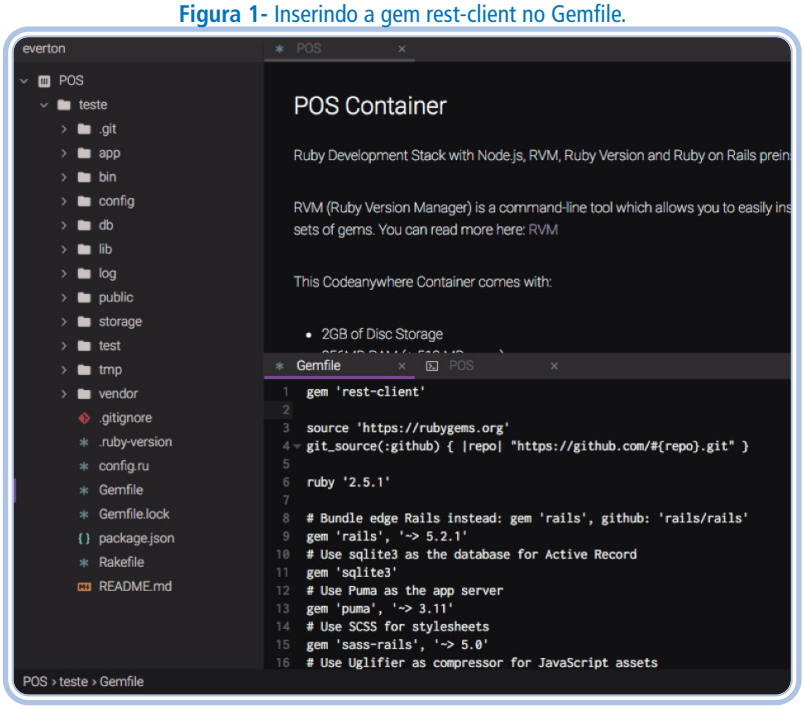
Pronto, agora, vamos criar o Gemfile do nosso projeto executando o comando:

```
bundle init
```

Criado o nosso Gemfile, abra-o e inclua uma linha nele, conforme a Figura 1:

```
gem 'rest-client'
```

Figura 1- Inserindo a gem rest-client no Gemfile.



Fonte: acervo pessoal.

Salve o arquivo e, no terminal de comandos, digite:

```
bundle install
```

Pronto, a **gem rest-client** está instalada em nosso projeto.

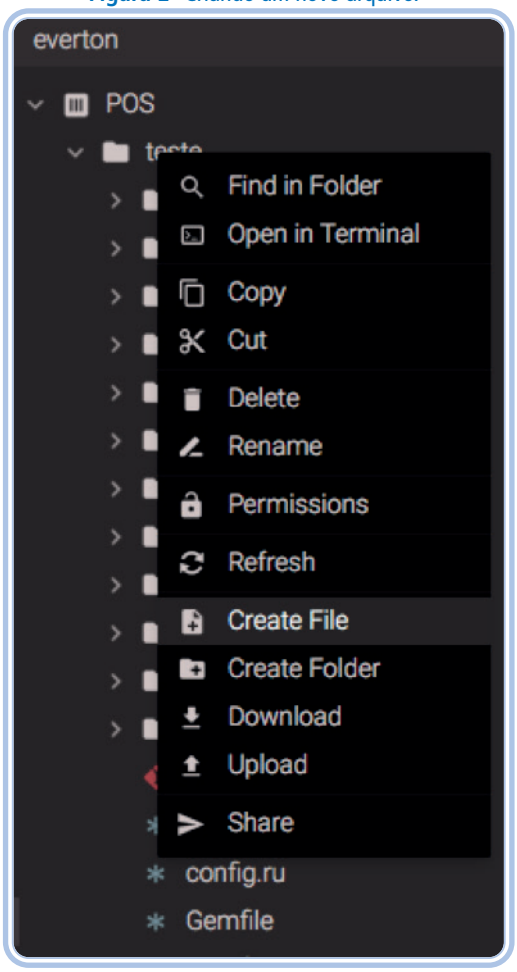
[**SAIBA MAIS!** Mas o que é o rest-client? Abra o navegador no endereço https://github.com/rest-client/rest-client.

A gem RestClient é um cliente REST/HTTP a qual irá nos ajudar a acessar APIs REST de forma fácil e rápida, a partir de alguns exemplos de como enviar requisições, usando diferentes métodos HTTP.

O ViaCEP é um webservice gratuito, de alto desempenho para consultar Códigos de Endereçamento Postal (CEP) do Brasil. Essa é a API a ser utilizada nesta aula, a fim de ilustrar como consumir serviços REST através da linguagem Ruby. Caso necessário, podemos acessar todos os detalhes de como utilizar essa API no endereço https://viacep.com.br/.

Vamos voltar ao nosso projeto, no Codeanywhere, e criar um script para acessar o serviço do ViaCEP. Crie um script Ruby clicando com o botão direito na pasta do projeto, depois clique em Create File (Figura 2) e, em seguida, nomeie o arquivo com cep.rb.

Figura 2- Criando um novo arquivo.



Fonte: acervo pessoal.

Nesse script, insira o seguinte código:

```
require 'rest-client'
require 'json'
```

```
puts “Digite um CEP:”
cep = gets.chomp

res = RestClient.get “https://viacep.com.br/ws/#{cep}/json/”

endereco = JSON.parse(res.body)

puts endereco
```

Inicialmente, fizemos o require ‘rest-client’ para o nosso script ter acesso à gem instalada. Depois, utilizamos require ‘json’ para poder usar a classe JSON, que irá nos auxiliar a tratar as respostas HTTP, as quais estarão formatadas em JSON.

Solicitamos que o usuário digite um CEP, e armazenamos o valor informado na variável cep. Sucessivamente, usamos o método get da classe RestClient para enviar uma requisição ao serviço do ViaCEP. Perceba que, no endereço do serviço, estamos passando o CEP informado pelo usuário, através da variável #{cep}.

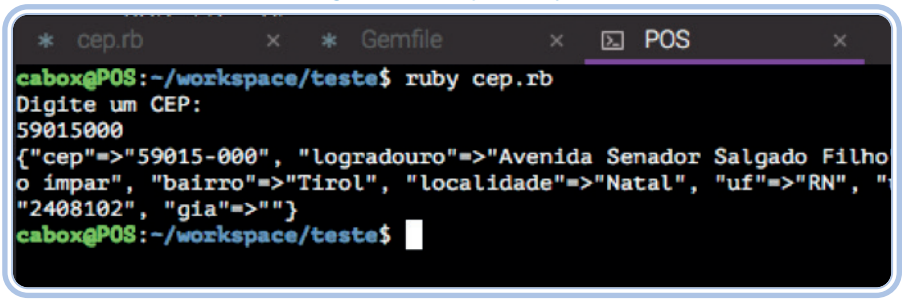
A resposta HTTP enviada pelo serviço será armazenada na variável res. O corpo da resposta, então, é tratado pelo método parse da classe JSON, que transforma a String contida no corpo da resposta em um hash em Ruby. Dessa forma, a variável endereco contém um hash em Ruby com o endereço do CEP informado pelo usuário. Por fim, a variável endereco é apresentada ao usuário.

Agora, vamos usar o nosso script cep.rb. No Codeanywhere, abra o terminal e digite:

```
ruby cep.rb
```

Como exemplo, utilizaremos o CEP do IFRN Campus Natal - Zona Leste: 59015000. Veja o resultado na Figura 3.

Figura 3 - Execução do cep.rb



Fonte: acervo pessoal.

Pronto, nosso script está funcionando! Obtivemos como saída um objeto JSON, com as chaves: cep, logradouro, complemento, bairro, localidade, uf, unidade, ibge e gia. Esse é mais um exemplo de API, o qual não nos exigiu nenhum tipo de autenticação para poder utilizá-la.

ATIVIDADE

1) Siga as orientações desta aula, crie o script cep.rb e execute-o inserindo o CEP de onde você reside.

Novamente, utilize a API do SUAP para ilustrar como se acessa uma API utilizando um token através da linguagem Ruby.

Crie um novo script, chamado suap.rb, e utilize o seguinte código:

```
require 'rest-client'
require 'json'

puts "Digite sua matrícula:"
matricula = gets.chomp

puts "Digite sua senha do SUAP:"
senha = gets.chomp

body = {
   "username": matricula,
   "password": senha
}
```

```
headers = {
   content_type: :json
}

resposta = RestClient.post "https://suap.ifrn.edu.br/api/v2/
autenticacao/token/", body.to_json, headers
json = JSON.parse(resposta.body)
token = json["token"]

headers = {
   :Authorization => "JWT #{token}"
}

resposta = RestClient.get "https://suap.ifrn.edu.br/api/v2/minhas-
informacoes/meus-dados", headers
json = JSON.parse(resposta.body)

puts json

endereco
```

Nas primeiras linhas, use o require para utilizar o rest-client e a classe JSON. Em seguida, solicitamos que digite nome de usuário e senha do SUAP, para poder usar o serviço de autenticação do SUAP e, assim, requerer seu token.

Depois, criamos a variável body e guardamos um hash, o qual contém as credenciais do usuário, login e senha. Também criamos a variável headers que também contém um hash com o Content-Type da nossa requisição.

Em sequência, usamos o método post da classe RestClient para enviar uma requisição HTTP POST. Para esse método, estamos passando três parâmetros: o endereço do serviço de autenticação do SUAP, o corpo da requisição, que é o hash contido na variável body no formato JSON, e o cabeçalho da requisição.

Assim como fizemos no script anterior, estamos transformando o corpo da resposta em um hash e guardando na variável json. Em seguida, acessamos a posição "token" do hash e guardamos o valor na variável token. Temos, então, o token de acesso do usuário.

Agora, estamos redefinindo a variável headers para usarmos o token no cabeçalho da requisição. Em seguida, usamos o método get da classe RestClient para enviar uma requisição HTTP GET ao serviço que recupera os dados do usuário no SUAP. Para esse método, passamos o endereço do serviço e o cabeçalho, o qual contém o token de acesso. Novamente, a resposta é tratada pelo método parse da classe JSON e, então, apresentamos o resultado para o usuário.

Teste o script digitando no terminal:

```
ruby suap.rb
```

Digite suas credenciais (login e senha) e observe que seus dados do SUAP serão apresentados no terminal.

ATIVIDADE

2) Siga as orientações da aula até aqui, crie o script suap.rb e execute-o inserindo suas credenciais de acesso ao SUAP.

VIDEOAULA

- Consumindo serviços com um cliente REST. Disponível em: https://youtu.be/TWfmXruzmPI.

RESUMINDO

Nesta Aula 7, aprendemos a consumir serviços utilizando a linguagem Ruby. Para isso, usamos o ambiente de desenvolvimento Codeanywhere e, através de dois exemplos, implementamos dois scripts Ruby e consumimos serviços utilizando REST e JSON.

Aula 8 - Oferecendo serviços com Rails

Até o momento, estudamos como consumir serviços utilizando a linguagem Ruby. Nesta Unidade VIII, iremos aprender a oferecer serviços utilizando o framework Ruby on Rails. Exemplificaremos com uma calculadora simples, a qual faz algumas operações matemáticas.

Portanto, iremos construir uma aplicação em Ruby on Rails, no Codeanywhere, e oferecer uma API com dois serviços, quais sejam: somar (para somar dois números) e fatorial (para calcular o fatorial de um número). Em ambos, os valores serão fornecidos para eles através do corpo de uma requisição HTTP POST.

Objetivo de aprendizagem

- Aprender a oferecer serviços utilizando o framework Ruby on Rails.

Desenvolvendo o conteúdo

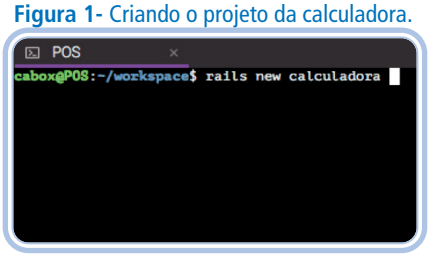
Desenvolvimento da calculadora

O desenvolvimento da calculadora ocorre em quatro etapas. Nas próximas seções, veremos cada uma.

Criando o projeto no Codeanywhere

Para criar seu projeto, entre na sua conta do Codeanywhere e acesse o container para projetos **Ruby on Rails**, o qual criamos na Aula 1. No terminal do Codeanywhere, digite o comando apresentado na Figura 1.

Figura 1- Criando o projeto da calculadora.



Fonte: acervo pessoal.

Uma vez criado o projeto, precisamos estar dentro do seu respectivo diretório para realizar as devidas alterações. Por essa razão, é importante se certificar de que você está dentro do diretório "calculadora" para ter êxito nos sucessivos procedimentos.

Desenvolvendo o controlador

Tendo criado o projeto, vamos criar um controlador chamado Calc, o qual irá conter os métodos que irão atender às requisições da nossa API. No terminal de comandos do Codeanywhere, execute o comando

```
rails generate controller calc
```

Esse comando irá gerar um controlador chamado CalcController em app/controllers/calc_controller.rb. Abra-o e insira o código presente na Figura 2.

Figura 2- Código do calc_controller.rb

```
calc_controller.rb
1  class CalcController < ApplicationController
2    skip_before_action :verify_authenticity_token
3
4    def somar
5      render json: { resultado: params[:n1] + params[:n2] }
6    end
7
8    def fatorial
9      fatorial = 1
10     numero = params[:numero]
11     for i in (1..numero)
12       fatorial *= i
13     end
14     render json: { resultado: fatorial }
15   end
16 end
17
```

Fonte: acervo pessoal.

Analisando a Figura 2, observe que, na linha 2, usamos skip_before_action :verify_authenticity_token. Isso é necessário para que o framework desabilite a proteção a ataques de Cross-site Request Forgery (CSRF) a esse controlador. Caso essa linha não seja incluída, não conseguiremos enviar requisições HTTP POST a esse controlador.

[**SAIBA MAIS** - Para saber mais sobre CSRF, acesse: https://pt.wikipedia.org/wiki/Cross-site_request_forgery.

Em seguida, há a implementação de dois métodos: somar e fatorial.

No método somar, temos a chamada do método render, que é usado para apresentar uma resposta ao usuário. Nesse caso, estamos apresentando um resultado em JSON. Em seguida, temos um hash com um único par, o qual a chave é resultado e o valor é o resultado da soma entre params[:n1] e params[:n2], valores fornecidos pelo usuário para que sejam somados.

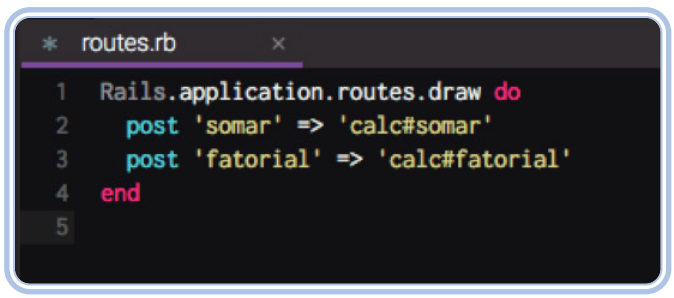
Já a implementação do método fatorial começa com a definição da variável fatorial com valor 1. Em sequência, guardamos o valor de params[:numero] na variável numero; params[:numero] é o valor fornecido pelo usuário.

Entre as linhas 11 e 13, há um laço que calcula o fatorial do valor contido em numero. Calculado o fatorial, o resultado é apresentado para o usuário na linha 14. Por fim, salve esse script.

Configurando as rotas

Com o controlador implementado, configure as rotas da nossa aplicação. Para tanto, abra o arquivo de rotas config/routes.rb e aplique o código apresentado na Figura 3.

Figura 3 - Arquivo de configuração de rotas.



Fonte: acervo pessoal.

Nessa Figura 3, na linha 2, configuramos o método somar do controlador calc para responder às requisições POST enviadas para o endereço /somar; e, na linha 3, configuramos o método fatorial do controlador calc para responder às requisições POST enviadas para o endereço /fatorial. Feito isso, salve o script.

ATIVIDADES

1) Siga as orientações desta aula, gere e configure o controlador Calc e as rotas da nossa aplicação.

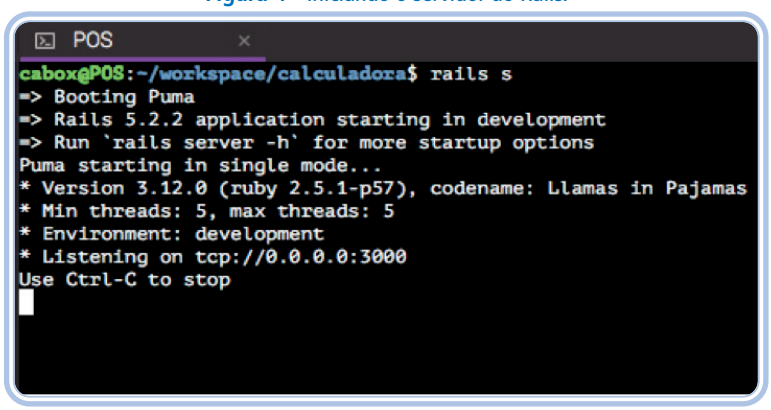
Testando nossa aplicação

O primeiro passo é executar a aplicação Ruby on Rails. Para isso, basta acessar o terminal do Codeanywhere, dentro da pasta do nosso projeto calculadora e executar o comando:

```
rails s
```

Logo após, nossa aplicação estará em execução; assim, o servidor da nossa aplicação ficará aguardando as requisições, como na Figura 4.

Figura 4 - Iniciando o servidor do Rails.



Fonte: acervo pessoal.

Iniciado o servidor da nossa aplicação calculadora, vamos, inicialmente, testar o serviço somar. Para isso, abra o Advanced Rest Client (ARC), configure o método POST e a URL https://pos-evertonfca55297.codeanyapp.com/somar. Verifique a URL do endereço da aplicação, assim como descrevemos na Aula 1. Este deve ser seguido do /somar, para esse caso.

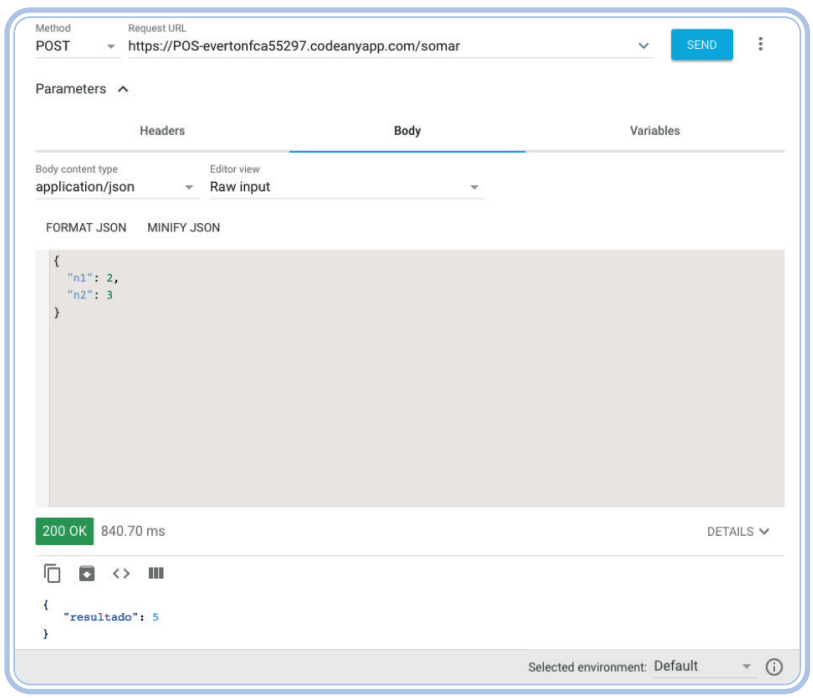
Ainda no ARC, na aba Headers, adicione um cabeçalho com o Header name: Content-Type e o Header value: application/json. Clique na aba Body e adicione o seguinte corpo em JSON:

```
{
  “n1”: 2,
  “n2”: 3
}
```

Lembre-se: os valores de n1e n2 são aqueles que desejamos que o nosso serviço retorne a sua some. Feito isso, clique em Send. A resposta da nossa aplicação deverá ser o exibida na Figura 5.

```
{
  “resultado”: 5
}
```

Figura 5 - Resultado do serviço somar no ARC.



Fonte: acervo pessoal.

Caso tenha recebido esse resultado, parabéns! Você acabou de construir e de consumir seu próprio serviço.

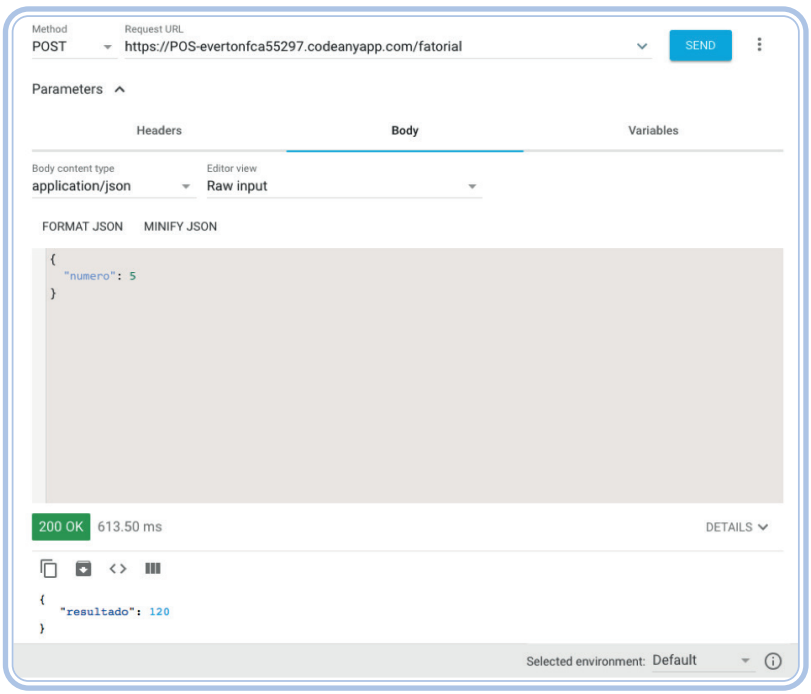
Agora, vamos testar o serviço que calcula o fatorial. Para tal, devemos alterar a URL para https://pos-evertonfca55297.codeanyapp.com/fatorial. Novamente, verifique o endereço da sua aplicação, assim como apresentamos em nossa primeira aula, e insira /fatorial ao final. Modifique o corpo da requisição para o código em JSON no ARC:

```
{
  “numero”: 5
}
```

Lembre-se de que o valor de numero é aquele que nós desejamos calcular o seu fatorial. Sequencialmente, clique em Send. Como resposta, temos o valor referente ao fatorial do número 5 (Figura 6).

```
{
  “resultado”: 120
}
```

Figura 6 - Resultado do serviço fatorial no ARC.



Fonte: acervo pessoal.

Caso tenha recebido a resposta acima, significa que o nosso serviço está funcionando corretamente. Parabéns!

2) Siga as orientações desta aula e consuma os serviços somar e fatorial da nossa aplicação pelo ARC.

RESUMINDO

Nesta Aula 8, foram apresentadas algumas funções e recursos do Codeanywhere. Aprendemos a usar o terminal de comandos e a compartilhar seu containers e conhecemos o seu dashboard. Dessa forma, concluímos a abordagem sobre o Codeanywhere.

AVALIAÇÃO DE APRENDIZAGEM

1) Após estudar o conteúdo desta aula, desenvolva o serviço de subtrair que deve retornar o resultado da subtração entre 2 valores. Esse novo serviço deverá compor o nosso projeto calculadora e deve atender a requisições POST.

Aula 9 - OAuth 2.0

Você já deve ter se deparado com a seguinte situação: estava navegando na Internet e descobriu um novo site ou serviço do qual gostou, e tem interesse em se cadastrar nele. Esse site, então, oferece a possibilidade de, ao invés de preencher um formulário de cadastro, se cadastrar utilizando suas informações de um outro site/serviço.

Como exemplo disso, temos o Spotify https://spotify.com, um serviço de transmissão de músicas. Ao entrar no formulário de cadastro do Spotify, ilustrado na Figura 1, você tem a opção de se inscrever usando seus dados do Facebook.

Objetivos de aprendizagem

- Conhecer o protocolo OAuth 2.0.
- Implementar o protocolo OAuth 2.0 com o framework Ruby on Rails.

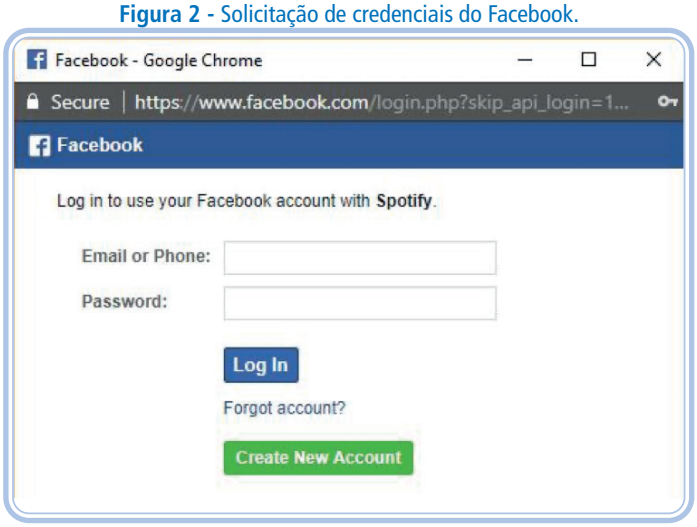
Figura 1- Formulário de cadastro do Spotify.

Como isso é possível? Seus dados do Facebook são públicos para que outros serviços possam utilizar? De que forma isso afeta a sua privacidade?

Na realidade, isso é possível devido a um protocolo de autorização chamado OAuth [1], que, atualmente, está em sua versão 2.0.

O OAuth permite que você (usuário) autorize aplicações a utilizarem seus dados e permissões em outras aplicações. No exemplo citado, você poderia autorizar o Spotify a utilizar seus dados do Facebook. Ao clicar no botão "Inscrever-se com o Facebook" (Figura 1), suas credenciais deste site seriam solicitadas, conforme ilustrado na Figura 2.

Figura 2 - Solicitação de credenciais do Facebook.



Com as credenciais fornecidas, você, enquanto usuário, poderia dar permissões para o Spotify acessar seus dados ou executar ações dentro do Facebook em seu nome.

A Figura 3 representa o Spotify solicitando permissão para publicar no seu perfil do Facebook.

Figura 3 - Permissão para publicar no Facebook.

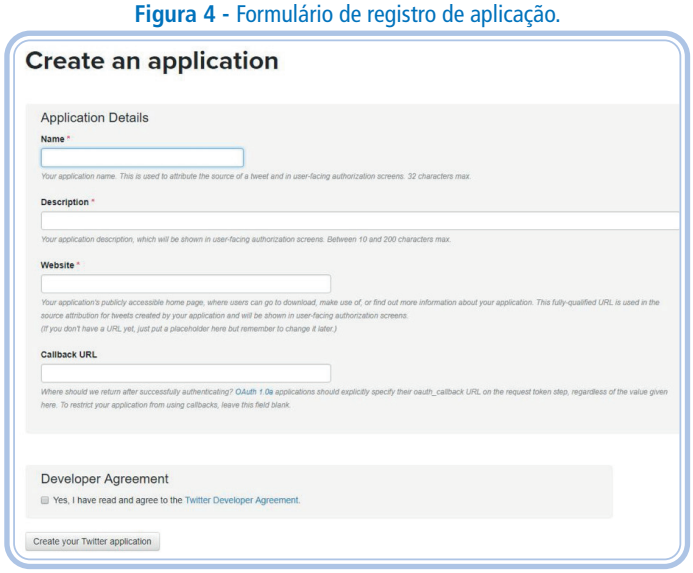
Autenticação com OAuth 2.0

Para apresentar a autenticação através de OAuth, vamos desenvolver uma aplicação em Ruby on Rails na qual usuários do twitter possam usá-la para buscar tweets com uma determinada hashtag e postar tweets.

Registrando sua aplicação

Antes de começarmos o desenvolvimento da nossa aplicação, é preciso ter uma conta no Twitter e a nossa aplicação registrada. Caso você não tenha, faça uma através do endereço https://twitter.com/signup. Para registrar a nossa aplicação, acesse o endereço https://apps.twitter.com e clique em Create new App.

Figura 4 - Formulário de registro de aplicação.



Preencha o formulário acima com os dados:

- Name (nome): nome da aplicação, de sua escolha (precisa ser único).
- Description (descrição): preencha com "Aprendendo a desenvolver aplicações usando OAuth" (sem aspas).
- Website: seu website (pode ser fictício).
- Callback URL: deixe em branco.
- Marque o checkbox "Yes, I have read and agree to the Twitter Developer Agreement".
- Clique no botão "Create your Twitter application".

Feito isso, sua aplicação estará registrada no Twitter. Clique na aba Keys and Access Tokens e, nessa página, duas informações são importantes: Consumer Key (API Key) e Consumer Secret (API Secret). O Consumer Key é um identificador para a sua aplicação, e essa informação pode ser pública. O Consumer Secret é uma informação secreta e, portanto, deve ser mantida em sigilo.

Para a nossa aplicação poder usar a nossa conta do Twitter, clique no botão Create my access token, o qual aparece no final da página (Figura 5).

Figura 5 - Criando seu access token.

Feito isso, mantenha essa aba do seu navegador aberta, pois, em breve, precisaremos do Consumer Key, Consumer Secret, Access Token e do Access Token Secret.

Desenvolvendo a aplicação

Agora, iniciemos o desenvolvimento da aplicação. Acesse o Codeanywhere, abra nosso container e crie um novo projeto Ruby on Rails chamado twitter. Para isso, no terminal do Codeanywhere, digite o comando:

```
rails new twitter
```

Criado o projeto, precisamos estar dentro do seu respectivo diretório para realizar as devidas alterações. Por isso, é importante se certificar de que você está, neste caso, dentro do diretório twitter, para que consigamos executar as próximas instruções.

Em seguida, instale o gerenciador de dependências Bundler ao nosso projeto usando, no terminal, este comando:

```
gem install bundler
```

Com o Bundler instalado, gere o Gemfile no nosso projeto executando, no terminal, o comando:

```
bundle init
```

O comando acima irá gerar o arquivo Gemfile no seu projeto. Abra esse arquivo e adicione esta linha nele (no final):

```
gem 'twitter
```

Salve o Gemfile e execute o comando abaixo no terminal, para instalar a gem Twitter ao projeto.

```
bundle install
```

Em seguida, vamos criar um arquivo de configuração para guardar nossas chaves (Keys), geradas quando cadastramos nossa aplicação no Twitter. Para isso, crie um novo arquivo no projeto chamado config.yml, e, nele, cole o conteúdo:

```
consumer_key: "COLE AQUI SEU CONSUMER KEY"
consumer_secret: "COLE AQUI SEU CONSUMER SECRET"
access_token: "COLE AQUI SEU ACCESS TOKEN"
access_token_secret: "COLE AQUI SEU ACCESS TOKEN SECRET"
```

Não esqueça de fazer as devidas substituições no config.yml e, depois, salvar o arquivo.

Sucessivamente, crie um script, chamado buscar.rb, no projeto e use o código:

```
require 'twitter'
require 'yaml'


yaml = YAML.load_file('config.yml')

client = Twitter::REST::Client.new do |config|
 config.consumer_key        = yaml['consumer_key']
 config.consumer_secret     = yaml['consumer_secret']
```

```
  config.access_token       = yaml['access_token']
  config.access_token_secret = yaml['access_token_secret']
end

client.search("#ifrn", result_type: "recent").each do |tweet|
  puts tweet.text
end
```

Inicialmente, fizemos o require 'twitter' para o nosso script ter acesso à gem que instalamos. Em seguida, utilizamos require 'yaml' para poder usar a classe YAML, que permite importar os dados do arquivo config.yml.

Logo, é realizada uma busca por novas mensagens no Twitter, através da hashtag #ifrn. Caso exista uma nova mensagem no tempo da execução do nosso arquivo, o seu texto será impresso na tela.

Agora, vamos usar o nosso script buscar.rb. No Codeanywhere, abra o terminal e digite:

```
ruby buscar.rb
```

Pronto, nosso script está funcionando! Caso alguma nova mensagem seja postada no Twitter, esta será impressa na tela.

ATIVIDADE

1) Siga as orientações desta aula e crie o projeto twitter.

RESUMINDO

Nesta aula, tratamos do oAuth 2.0, um serviço de autenticação bastante utilizado pelos serviços oferecidos na Internet. Criamos uma aplicação que utiliza o

Referências

Advanced REST Client. **Free and open source API testing tool**. Disponível em: https://install.advancedrestclient.com/install. Acesso em: 02 de ago. 2018.

API do SUAP. Disponível em: <https://suap.ifrn.edu.br/api/docs>. Acesso em: 02 de ago. de 2018.

CODEANYWHERE. ***Cross platform cloud ide.*** Disponível em: <https://codeanywhere.com/>. Acesso em: 02 de nov. de 2018.

Deivid Fortuna. **FIPE API HTTP REST**. Disponívem em: <https://deividfortuna.github.io/fipe/>. Acesso em: 12 de ago. de 2018.

Geeks for Geeks. **Difference between JSON and XML**. Disponível em: <https://www.geeksforgeeks.org/difference-between-json-and-xml/>. Acesso em: 02 de ago. de 2018.

GORAILS. ***Setup ruby on rails on ubuntu 16.10 yakkety yak***. Disponível em: <https://gorails.com/setup/ubuntu/16.10>. Acesso em: 12 de nov. de 2018.

iSET. **O que é API RESTful?** Entenda aqui! Disponível em: <https://blog.iset.com.br/o-que-e-api-restful-entenda-aqui/>. Acesso em: 12 de ago. de 2018.

JSON. **Introdução ao JSON**. Disponível em: <https://www.json.org/json-pt.html>. Acesso em: 02 de ago. de 2018.

MDN web doc. **Códigos de status de respostas HTTP**. Disponível em: <https://developer.mozilla.org/pt-BR/docs/Web/HTTP/Status>. Acesso em: 02 de set. de 2018.

MuleSoft. **What is an API? (Application Programming Interface).** Disponível em: <https://www.mulesoft.com/resources/api/what-is-an-api>. Acesso em: 12 de ago. de 2018.

Nando Vieira. **Entendendo um pouco mais sobre o protocolo HTTP**. Disponível em: <https://nandovieira.com.br/entendendo-um-pouco-mais-sobre-o-protocolo-http>. Acesso em: 02 de set. de 2018.

OAUTH 2.0. ***An open protocol to allow secure authorization in a simple and standard method from web, mobile and desktop applications***. Disponível em: <https://oauth.net>. Acesso em: 10 de out. de 2018.

RUBY. ***Ruby programming language***. Disponível em: <http://ruby-lang.org/>. Acesso em: 29 de dez. de 2017.

RUBY ON RAILS. ***Ruby on rails*** | a web-application framework that includes everything

needed to create database-backed web applications according to the model-view-controller (mvc) pattern. Disponível em: <http://rubyonrails.org/>. Acesso em: 10 de out. de 2018.

TWITTER. ***Tweets, optimize ads, and create unique customer experiences***. Disponível em: <https://apps.twitter.com>. Acesso em: 02 de set. de 2018.

UBUNTU. ***The leading operating system for pcs, iot devices, servers and the cloud | ubuntu***. Disponível em: <https://www.ubuntu.com/>. Acesso em: 02 de nov. de 2018.

VIACEP. ***Consulte CEPs de todo o Brasil***. Disponível em: <https://viacep.com.br.> Acesso em: 02 de dez. de 2018.

World Wide Web Consortium (W3C). **HTTP - Hypertext Transfer Protocol**. Disponível em: <https://www.w3.org/Protocols/>. Acesso em: 02 de set. de 2018.

INSTITUTO FEDERAL
Rio Grande do Norte
Campus Avançado Natal - Zona Leste