Consumindo API com Elixir

Vinicius Negreiros de Melo

API CoinLore

- Simples e útil
- interesse pessoal
- Dados concretos


```
{
  "data": [
    {
      "id": "90",
      "symbol": "BTC",
      "name": "Bitcoin",
      "nameid": "bitcoin",
      "rank": 1,
      "price_usd": "6456.52",
      "percent_change_24h": "-1.47",
      "percent_change_1h": "0.05",
      "percent_change_7d": "-1.07",
      "price_btc": "1.00",
      "market_cap_usd": "111586042785.56",
      "volume24": 3997655362.9586277,
      "volume24a": 3657294860.710187,
      "csupply": "17282687.00",
      "tsupply": "17282687",
      "msupply": "21000000"
    },
  "info": {
    "coins_num": 1969,
    "time": 1538560355
  }
}
```

Função Main()

```elixir
def main([rank_str]) do
  rank = String.to_integer(rank_str)
  CoinloreApi.obtem_moedas()
  |> mostra_resultado(rank)
end
```

Voltar ao índice

Request da API

```elixir
defmodule CoinloreApi do
  @url "https://api.coinlore.net/api/tickers/"

  #função que fará o todo o processo de requisição da API
  def obtem_moedas() do
    HTTPoison.get(@url)
    |> processa_resposta
  end

  #função que processará a resposta tendo como primeira cláusula o cenário em que a API retorna o
  defp processa_resposta({ :ok, %HTTPoison.Response{status_code: 200, body: b}}) do { :ok, b}
  #Retorna "ok" mostrando que deu tudo certo e define a resposta da API como "b"
  end

  #segunda cláusula onde a requisição falhou e retorna a tupla error e r sendo r a razão do erro
  defp processa_resposta({ :error, r}), do: { :error, r}

  #terceira cláusula trata o cenário que a API responde mas a resposta é diferente do codigo 200
  defp processa_resposta({ :ok, %HTTPoison.Response{ status_code: _, body: b }}) do {:error, b}
  #também retorna error mas ao invés de r, retorna oque foi recebido pela API
  end
end
```

Voltar ao índice

```elixir
# Função que mostra o resultado, verificando se houve erro na resposta da API
defp mostra_resultado({:error, _}, _rank) do
  IO.puts "Ocorreu um erro"
end

# Função que decodifica o JSON recebido da API
defp mostra_resultado({:ok, json}, rank) do
  case Poison.decode(json) do
    {:ok, %{"data" => moedas}} ->
      # Exibe a lista completa de moedas
      mostra_moedas(moedas)
      IO.puts "Agora exibindo os detalhes da moeda de rank #{rank}:"
      # Exibe os detalhes da moeda com o rank fornecido
      mostra_moeda_por_rank(moedas, rank)

    {:error, reason} ->
      IO.puts "Erro ao decodificar JSON: #{reason}"

    _ ->
      IO.puts "Formato inesperado de resposta"
  end
end
```

Mostra_resultado

Exibindo o top 100

```elixir
# Função para exibir todas as moedas
defp mostra_moedas([]), do: IO.puts "Abaixo listaremos os detalhes da moeda selecionada"
defp mostra_moedas([i | resto]) do
  name = i["name"]
  rank = i["rank"]
  price_usd = i["price_usd"]
  IO.puts "#{rank} | #{name} | #{price_usd}"
  mostra_moedas(resto)
end
```

```elixir
# Função para exibir detalhes de uma moeda específica pelo rank
defp mostra_moeda_por_rank(moedas, rank) do
  case encontra_moeda(moedas, rank) do
    nil ->
      IO.puts "Nenhuma moeda encontrada com o rank #{rank}."
    moeda ->
      IO.puts "Detalhes da moeda #{moeda["name"]}:"
      IO.puts "Rank: #{moeda["rank"]}"
      IO.puts "Preço em USD: #{moeda["price_usd"]}"
      IO.puts "Capitalização de mercado: #{moeda["market_cap_usd"]}"
      IO.puts "Volume de 24h: #{moeda["volume24"]}"
  end
end
```

Função auxiliar encontra moeda

```elixir
# Função recursiva para encontrar a moeda com o rank especificado
defp encontra_moeda([], _rank), do: nil
defp encontra_moeda([moeda | resto], rank) do
  if moeda["rank"] == rank do
    moeda
  else
    encontra_moeda(resto, rank)
  end

end
```