IIC2113: Entrega 3

Repositorio Heredado: Exchange
Grupo 18

1. Escribir código funcional

```ruby
class PriceUpdateWorker
  include Sidekiq::Worker

  def perform(*args)
    @btf = Currency.find_by name: "BTF"
    @bth = Currency.find_by name: "BTH"
    btf_raise_value = rand >= 0.5
    bth_raise_value = rand >= 0.5
    if btf_raise_value
      @btf.current_value = @btf.current_value * 1.1
    else
      @btf.current_value = @btf.current_value * 0.9
    end

    if bth_raise_value
      @bth.current_value = @bth.current_value * 1.1
    else
      @bth.current_value = @bth.current_value * 0.9
    end
    @btf.save
    @bth.save
  end
end
```

```ruby
def fetch_transactions
    #hacer un try except para errores con el fetch
    #separar en distintos metodos para tener un buen diseño
    @account = Account.find_by(user_id: current_user.id)
    response = RestClient::Request.new(
        :method => :get,
        :url => 'https://5fc567594931580016e3bba4.mockapi.io/api/v1/transactions',
        :verify_ssl => false
     ).execute
    results = JSON.parse(response.to_str)
    results.each do |transaction|
        bank_transaction = BankTransaction.find_or_initialize_by(date: Time.at(transaction["createdAt"]
        if bank_transaction.rut == current_user.rut
            if bank_transaction.new_record?
                @account.update_attribute(:CLP, @account.CLP + bank_transaction.amount)
                bank_transaction.save!
            end
        end
    end
```

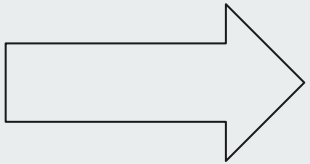
2. Refactorizar

```ruby
require 'sidekiq-scheduler'

class PriceUpdateWorker
  include Sidekiq::Worker

  def perform
    @btf = Currency.find_by name: "BTF"
    @bth = Currency.find_by name: "BTH"
    btf_raise_value = rand >= 0.5
    bth_raise_value = rand >= 0.5
    update(@btf, btf_raise_value)
    update(@bth, bth_raise_value)
  end

  def update(currency, value_rises)
    currency.current_value = if value_rises
                               currency.current_value * 1.1
                             else
                               currency.current_value * 0.9
                             end
    currency.save
  end
end
```

```ruby
def fetch_transactions
    # hacer un try except para errores con el fetch
    # separar en distintos metodos para tener un buen diseño
    @account = Account.find_by(user_id: current_user.id)

    begin
        response = RestClient::Request.new(
          :method => :get,
          :url => 'https://5fc567594931580016e3bba4.mockapi.io/api/v1/transactions',
          :verify_ssl => false
        ).execute
    rescue RestClient::ExceptionWithResponse => request_err
        return handle_bad_request(request_err, @account)
    rescue RestClient::ServerBrokeConnection => connection_err
        return handle_bad_connection(connection_err, @account)
    rescue RestClient::SSLCertificateNotVerified => secure_val_err
        return handle_bad_security_val(secure_val_err, @account)
    rescue => default_err
        return handle_error(default_err, @account)
    end

    results = JSON.parse(response.to_str)
    results.each do |transaction|
        bank_transaction = BankTransaction.find_or_initialize_by(date: Time.at(transaction["
```

Single - Responsibility - Principle

```ruby
class TransactionsController < ApplicationController
  def new
    @transaction = Transaction.new
  end

  def create
    @account = Account.find(transaction_params[:account_id])
    @currency = Currency.find(transaction_params[:currency_id])
    if BuySell.for(account: @account,
                   amount: transaction_params[:amount],
                   is_buy: transaction_params[:is_buy],
                   currency: @currency)
      @transaction = Transaction.new(transaction_params)
      @transaction.currency_price = @currency.current_value
      puts @transaction.inspect
      if @transaction.save
        UpdateValueCurrency.for(transaction: @transaction)
        UserNotifierMailer.send_transaction_email(@account.user).deliver
        redirect_to transactions_path
      else
        redirect_to transactions_path
      end
    else
      redirect_to transactions_path
    end
  end
```

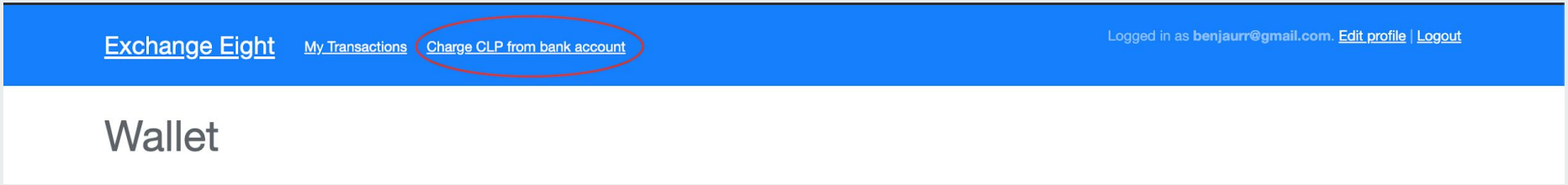


```ruby
require 'rest-client'
require 'json'

class BankTransactionsController < ApplicationController
    def new
        @bank_transaction = BankTransaction.new
    end

    def create
        @bank_transaction = BankTransaction.new(bank_transaction_params)
    end

    def fetch_transactions
        # hacer un try except para errores con el fetch
        # separar en distintos metodos para tener un buen diseño
        @account = Account.find_by(user_id: current_user.id)
```

Uso de Helper para función auxiliar

```ruby
module AccountsHelper
    def get_ammount_percentage(ammount, total)
        if total != 0
            return number_to_percentage((ammount / total) * 100, precision: 2)
        end
        return 0
    end
end
```

Lógica de carga de CLP


Exchange Eight
My Transactions
Charge CLP from bank account
Logged in as benjaurr@gmail.com. Edit profile | Logout
Wallet

Lógica de carga de CLP

```
    results.each do |transaction|
        bank_transaction = BankTransaction.find_or_initialize_by(date: Time.at(transaction["createdAt"])
        if bank_transaction.rut == current_user.rut
            if bank_transaction.new_record?
                @account.update_attribute(:CLP, @account.CLP + bank_transaction.amount)
                bank_transaction.save!
            end
        end
    end
    redirect_to @account
end
```

Manejo de errores

```
def fetch_transactions
    @account = Account.find_by(user_id: current_user.id)

    begin
        response = RestClient::Request.new(
        :method => :get,
        :url => 'https://5fc567594931580016e3bba4.mockapi.io/api/v1/transactions',
        :verify_ssl => false
      ).execute
    rescue RestClient::ExceptionWithResponse => request_err
        return handle_bad_request(request_err, @account)
    rescue RestClient::ServerBrokeConnection => connection_err
        return handle_bad_connection(connection_err, @account)
    rescue RestClient::SSLCertificateNotVerified => secure_val_err
        return handle_bad_security_val(secure_val_err, @account)
    rescue => default_err
        return handle_error(default_err, @account)
    end
```

Desafíos enfrentados

- Comprender lo recibido del proyecto *legacy*
- Comunicación para ir notificando cambios en el proyecto
- Automatización de procesos

IIC2113: Entrega 3

Repositorio Heredado: Exchange
Grupo 18