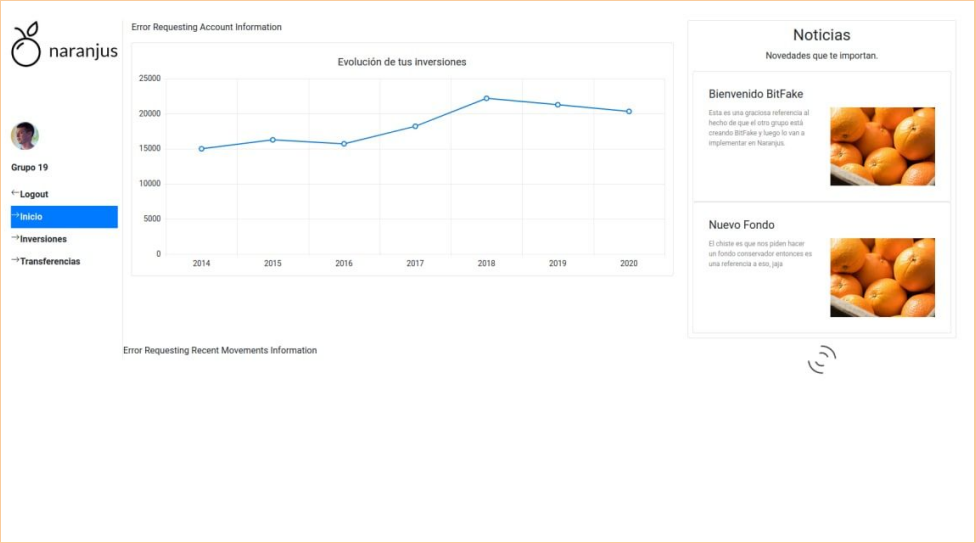
Grupo 19 - Diseño detallado de Software

naranjus

Grupo 19 - Entrega 2



Grupo 19 - Entrega 3

Diagramas usados

Diagrama de Arquitectura

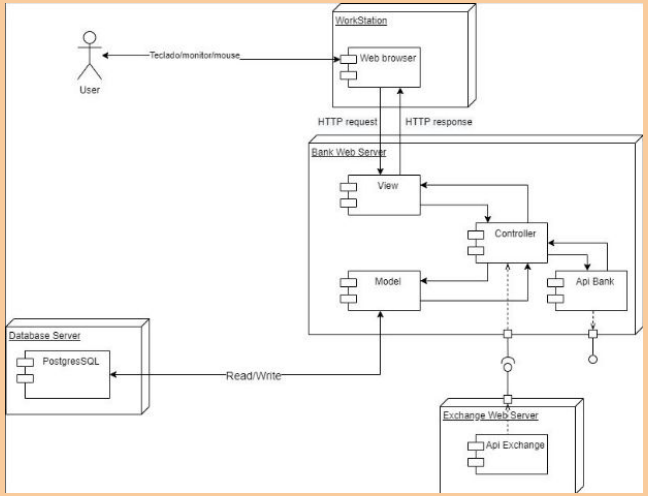
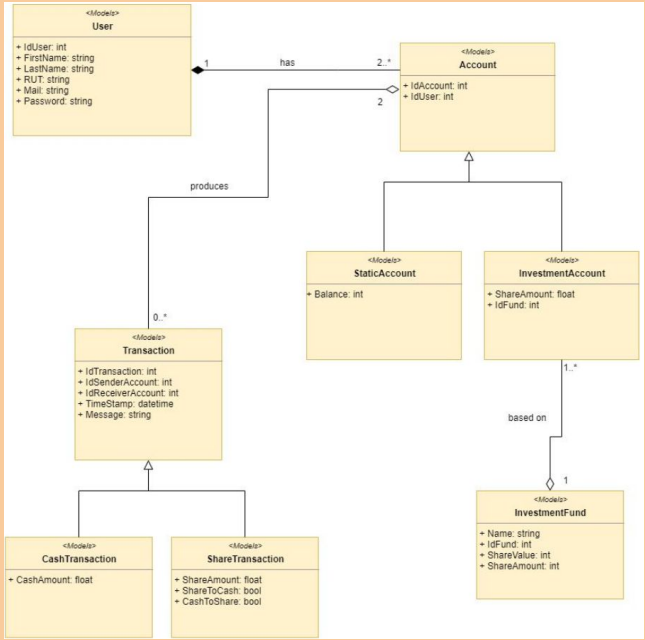
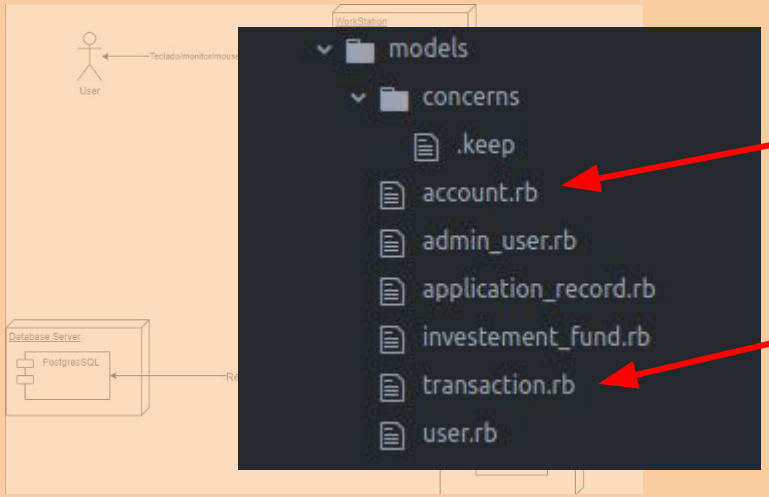
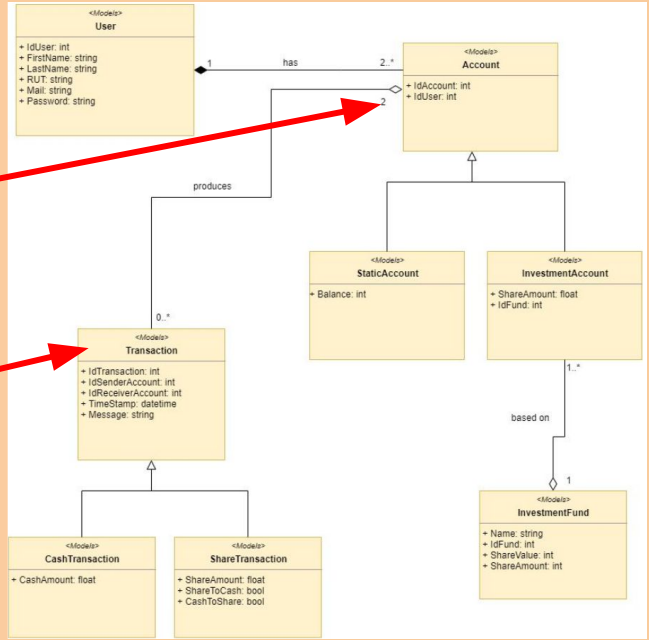


Diagrama de Clases

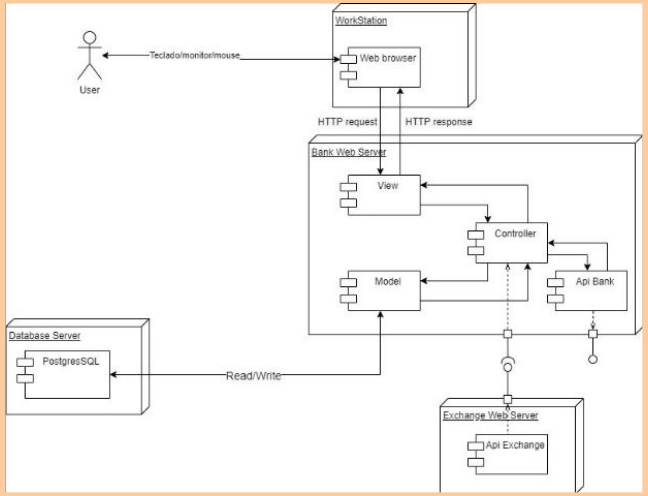


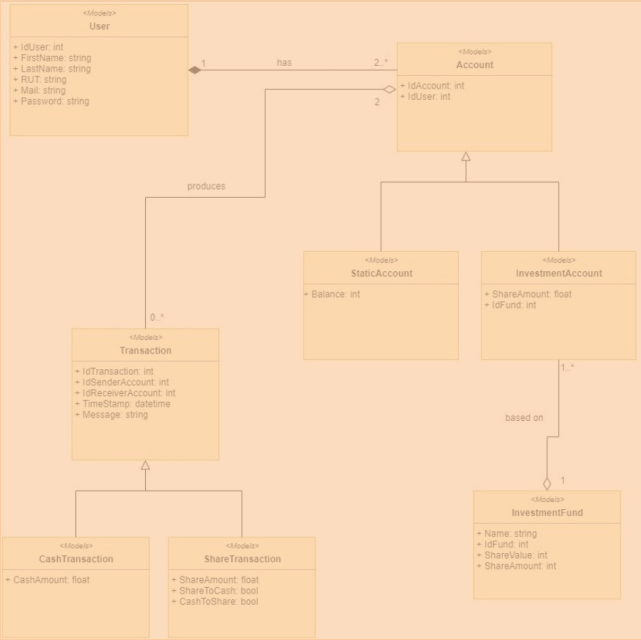
Diagramas usados



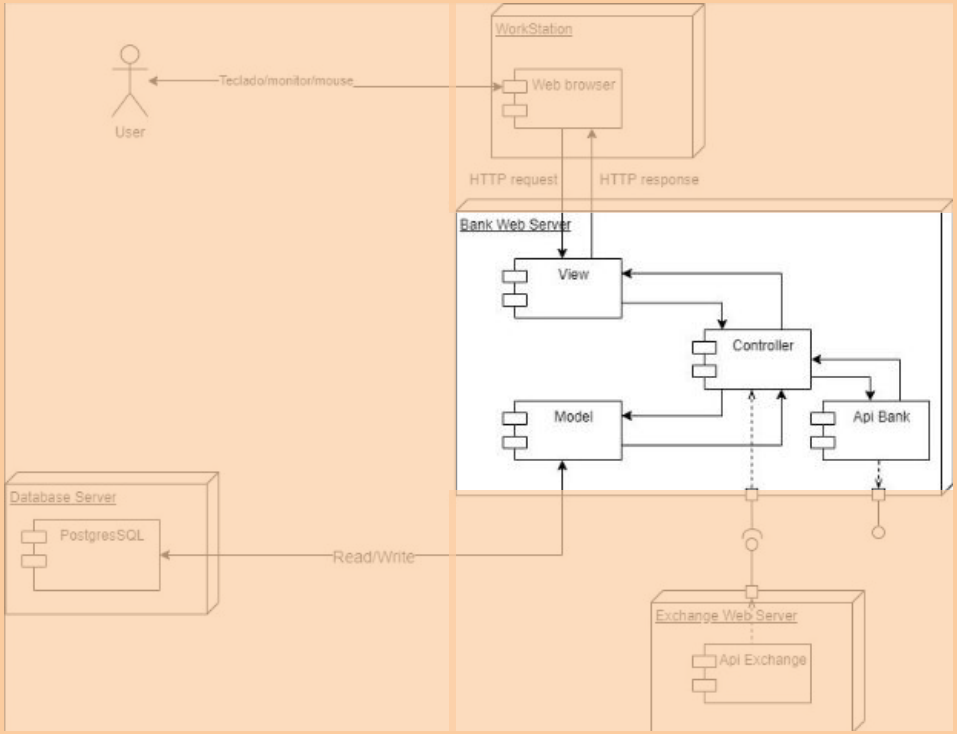


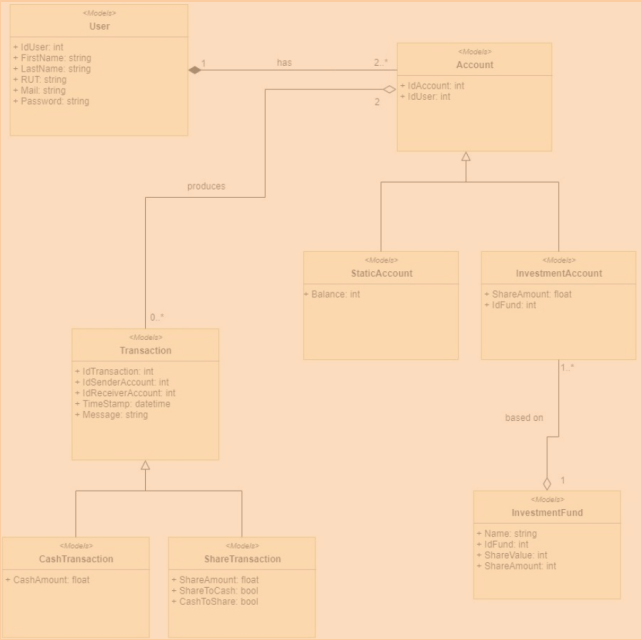
Diagramas usados





Diagramas usados





Ejemplos de buenas prácticas

Single-responsibility principle

TransactionHistory.tsx | Transfer.tsx | Login.tsx | Home.tsx

```
import React, { Component, Fragment } from "react";
import { Redirect } from "react-router-dom";
import Loading from "../layout/Loading";
import { isLogedIn, getHeaders } from '../services/isLogin';
import { transations_url } from '../services/api_url';

const fetch_state= {
  LOADING: 'loading',
  SUCCESS: 'success',
  AUTH_ERROR: 'auth_error',
  LOGIN_ERROR: 'login_error',
}

interface TransactionHistoryProps {
}

type transaction_person = {
  account_id: number,
  first_name: string,
  last_name: string,
}

type received_transaction_type = {
  id: number,
  message: string,
  cash_amount: number,
  updated_at: string,
  is_confirmed: boolean,
  sender: transaction_person,
};
```

Ejemplos de buenas prácticas

Single-responsibility principle

TransactionHistory.tsx | Transfer.tsx | Login.tsx | Home.tsx

```
1  import React, { Component, Fragment } from "react";
2  import { Redirect } from "react-router-dom";
3  import Loading from "../layout/Loading";
4  import { isLogedIn, getHeaders } from '../services/isLogin';
5  import { transations_url } from '../services/api_url';
6
7  const fetch_state= {
8    LOADING: 'loading',
9    SUCCESS: 'success',
10   AUTH_ERROR: 'auth_error',
11   LOGIN_ERROR: 'login_error',
12 }
13
14 interface TransactionHistoryProps {
15 }
16
17 type transaction_person = {
18   account_id: number,
19   first_name: string,
20   last_name: string,
21 }
22
23 type received_transaction_type = {
24   id: number,
25   message: string,
26   cash_amount: number,
27   updated_at: string,
28   is_confirmed: boolean,
29   sender: transaction_person,
30 };
31
```

Ejemplos de buenas prácticas

Api encapsulada

trxService.js

```javascript
const authHeaders = {
  "X-User-Email": localStorage.getItem("email"),
  "X-User-Token": localStorage.getItem("token"),
};

export async function getSharePrice() {
  const funds = await fetch(`${process.env.REACT_APP_SERVER_URL}/funds`, {
    headers: authHeaders,
  });
  const funds_json = await funds.json();
  let price = 0;
  if (funds_json && funds_json.length) {
    price = funds_json[0].share_value;
  }
  return price;
}

export async function getStaticAccount() {
  let accounts = await fetch(`${process.env.REACT_APP_SERVER_URL}/accounts/`, {
    headers: authHeaders,
  });
  accounts = await accounts.json();
  let account = accounts[0];
  accounts.forEach((acc) => {
    if (acc.is_static) account = acc;
  });
  account = await fetch(`${process.env.REACT_APP_SERVER_URL}/accounts/${account.id}`, {
    headers: authHeaders,
  });
  account = await account.json();
  return account;
}

export async function getInvestmentAccount() {
  let accounts = await fetch(`${process.env.REACT_APP_SERVER_URL}/accounts/`, {
    headers: authHeaders,
  });
  accounts = await accounts.json();
  let account = accounts[0];
  accounts.forEach((acc) => {
    if (!acc.is_static) account = acc;
  });
  account = await fetch(`${process.env.REACT_APP_SERVER_URL}/accounts/${account.id}`, {
    headers: authHeaders,
  });
  account = await account.json();
  return account;
}
```

Ejemplos de buenas prácticas

Api encapsulada

trxService.js

```javascript
const authHeaders = {
  "X-User-Email": localStorage.getItem("email"),
  "X-User-Token": localStorage.getItem("token"),
};

export async function getSharePrice() {
  const funds = await fetch(`${process.env.REACT_APP_SERVER_URL}/funds`, {
    headers: authHeaders,
  });
  const funds_json = await funds.json();
  let price = 0;
  if (funds_json && funds_json.length) {
    price = funds_json[0].share_value;
  }
  return price;
}

export async function getStaticAccount() {
  let accounts = await fetch(`${process.env.REACT_APP_SERVER_URL}/accounts/`, {
    headers: authHeaders,
  });
  accounts = await accounts.json();
  let account = accounts[0];
  accounts.forEach((acc) => {
    if (acc.is_static) account = acc;
  });
  account = await fetch(`${process.env.REACT_APP_SERVER_URL}/accounts/${account.id}`, {
    headers: authHeaders,
  });
  account = await account.json();
  return account;
}

export async function getInvestmentAccount() {
  let accounts = await fetch(`${process.env.REACT_APP_SERVER_URL}/accounts/`, {
    headers: authHeaders,
  });
  accounts = await accounts.json();
  let account = accounts[0];
  accounts.forEach((acc) => {
    if (!acc.is_static) account = acc;
  });
  account = await fetch(`${process.env.REACT_APP_SERVER_URL}/accounts/${account.id}`, {
    headers: authHeaders,
  });
  account = await account.json();
  return account;
}
```

Nos sentimos orgullosos de:

```
change-nine > exchange-nine > lib > tasks > update.rake
Pablo, a day ago | 1 author (Pablo)
MAX_VARIATION = 0.3      Pablo, 2 days ago • Finished with update prices

# Taken from here: https://stackoverflow.com/questions/1711681/how-to-best-create-a-random-float-in-a-range-between-two-float
def range (min, max)
  rand * (max-min) + min
end

def get_specific_variation(currency, price)
  selected_price = currency[price]
  new_price = range(
    selected_price*(1-MAX_VARIATION),
    selected_price*(1+MAX_VARIATION)
  )
  currency[price] = new_price
end

def zero_correction(currency, price)
  if currency[price] < 0
    currency[price] = 0
  end
end

def get_variation(currency)
  get_specific_variation(currency, "buy_price")
  get_specific_variation(currency, "sell_price")

  zero_correction(currency, "buy_price")
  zero_correction(currency, "sell_price")

  if currency.buy_price < currency.sell_price
    average = (currency.buy_price + currency.sell_price) / 2
    currency.buy_price = average
    currency.sell_price = average
  end
  return currency
end

namespace :update do
  desc "-------Update the price randomly-------"
  task :price => :environment do

    to_update_btf = Btf.first
    to_update_bth = Bth.first

    if !to_update_btf.nil? && !to_update_bth.nil?
      get_variation(to_update_btf)
      to_update_btf.save
```

```
end

namespace :update do
  desc "-------Update the price randomly-------"
  task :price => :environment do

    to_update_btf = Btf.first
    to_update_bth = Bth.first

    if !to_update_btf.nil? && !to_update_bth.nil?
      get_variation(to_update_btf)
      to_update_btf.save

      get_variation(to_update_bth)
      to_update_bth.save

      message = "---- Update Pricess ----\nBTF: buy:%f sell:%f\nBTH: buy:%f sell:%f\n" % [
        to_update_btf.buy_price,
        to_update_btf.sell_price,
        to_update_bth.buy_price,
        to_update_bth.sell_price
      ]
      Rails.logger.info message
    else
      Rails.logger.info "---- Update Pricess ----\nThere was an error in the currencies"
    end

  end

end
```

Nos sentimos orgullosos de:

```
change-nine > exchange-nine > lib > tasks > update.rake
Pablo, a day ago | 1 author (Pablo)
MAX_VARIATION = 0.3      Pablo, 2 days ago • Finished with update prices

# Taken from here: https://stackoverflow.com/questions/1711681/how-to-best-create-a-random-float-in-a-range-between-two-float
def range (min, max)
  rand * (max-min) + min
end

def get_specific_variation(currency, price)
  selected_price = currency[price]
  new_price = range(
    selected_price*(1-MAX_VARIATION),
    selected_price*(1+MAX_VARIATION)
  )
  currency[price] = new_price
end

def zero_correction(currency, price)
  if currency[price] < 0
    currency[price] = 0
  end
end

def get_variation(currency)
  get_specific_variation(currency, "buy_price")
  get_specific_variation(currency, "sell_price")

  zero_correction(currency, "buy_price")
  zero_correction(currency, "sell_price")

  if currency.buy_price < currency.sell_price
    average = (currency.buy_price + currency.sell_price) / 2
    currency.buy_price = average
    currency.sell_price = average
  end
  return currency
end

namespace :update do
  desc "-------Update the price randomly-------"
  task :price => :environment do

    to_update_btf = Btf.first
    to_update_bth = Bth.first

    if !to_update_btf.nil? && !to_update_bth.nil?
      get_variation(to_update_btf)
      to_update_btf.save
```

```
end

namespace :update do
  desc "-------Update the price randomly-------"

    if !to_update_btf.nil? && !to_update_bth.nil?
      get_variation(to_update_btf)
      to_update_btf.save

      get_variation(to_update_bth)
      to_update_bth.save

      message = "---- Update Pricess ----\nBTF: buy:%f sell:%f\nBTH: buy:%f sell:%f\n" % [
        to_update_btf.buy_price,
        to_update_btf.sell_price,
        to_update_bth.buy_price,
        to_update_bth.sell_price
      ]
      Rails.logger.info message
    else
      Rails.logger.info "---- Update Pricess ----\nThere was an error in the currencies"
    end

  end

end
```

Separamos las funciones

Nos sentimos orgullosos de:

```
change-nine > exchange-nine > lib > tasks > update.rake
Pablo, a day ago | 1 author (Pablo)
MAX_VARIATION = 0.3      Pablo, 2 days ago • Finished with update prices

# Taken from here: https://stackoverflow.com/questions/1711681/how-to-best-create-a-random-float-in-a-range-between-two-float
def range (min, max)
  rand * (max-min) + min
end

def get_specific_variation(currency, price)
  selected_price = currency[price]
  new_price = range(
    selected_price*(1-MAX_VARIATION),
    selected_price*(1+MAX_VARIATION)
  )
  currency[price] = new_price
end

def zero_correction(currency, price)
  if currency[price] < 0
    currency[price] = 0
  end
end

def get_variation(currency)
  get_specific_variation(currency, "buy_price")
  get_specific_variation(currency, "sell_price")

  zero_correction(currency, "buy_price")
  zero_correction(currency, "sell_price")

  if currency.buy_price < currency.sell_price
    average = (currency.buy_price + currency.sell_price) / 2
    currency.buy_price = average
    currency.sell_price = average
  end
  return currency
end

namespace :update do
  desc "-------Update the price randomly-------"
  task :price => :environment do

    to_update_btf = Btf.first
    to_update_bth = Bth.first

    if !to_update_btf.nil? && !to_update_bth.nil?
      get_variation(to_update_btf)
      to_update_btf.save
```

```
end

namespace :update do
  desc "-------Update the price randomly-------"

    if !to_update_btf.nil? && !to_update_bth.nil?
      get_variation(to_update_btf)

      message = "---- Update Pricess ----\nBTF: buy:%f sell:%f\nBTH: buy:%f sell:%f\n" % [
        to_update_btf.buy_price,
        to_update_btf.sell_price,
        to_update_bth.buy_price,
        to_update_bth.sell_price
      ]
      Rails.logger.info message
    else
      Rails.logger.info "---- Update Pricess ----\nThere was an error in the currencies"
    end

  end

end
```

Separamos las funciones

Programación defensiva

Nos sentimos orgullosos de:

```
change-nine > exchange-nine > lib > tasks > update.rake
Pablo, a day ago | 1 author (Pablo)
MAX_VARIATION = 0.3      Pablo, 2 days ago • Finished with update prices

# Taken from here: https://stackoverflow.com/questions/1711681/how-to-best-create-a-random-float-in-a-range-between-two-float
def range (min, max)
  rand * (max-min) + min
end

def get_specific_variation(currency, price)
  selected_price = currency[price]
  new_price = range(
    selected_price*(1-MAX_VARIATION),
    selected_price*(1+MAX_VARIATION)
  )
  currency[price] = new_price
end

def zero_correction(currency, price)
  if currency[price] < 0
    currency[price] = 0
  end
end

def get_variation(currency)
  get_specific_variation(currency, "buy_price")
  get_specific_variation(currency, "sell_price")

  zero_correction(currency, "buy_price")
  zero_correction(currency, "sell_price")

  if currency.buy_price < currency.sell_price
    average = (currency.buy_price + currency.sell_price) / 2
    currency.buy_price = average
    currency.sell_price = average
  end
  return currency
end

namespace :update do
  desc "-------Update the price randomly-------"
  task :price => :environment do

    to_update_btf = Btf.first
    to_update_bth = Bth.first

    if !to_update_btf.nil? && !to_update_bth.nil?
      get_variation(to_update_btf)
      to_update_btf.save
```

Separamos las funciones

Programación defensiva

Fácil de entender

Nos sentimos orgullosos de:

```
change-nine > exchange-nine > lib > tasks > update.rake
Pablo, a day ago | 1 author (Pablo)
MAX_VARIATION = 0.3      Pablo, 2 days ago • Finished with update prices

# Taken from here: https://stackoverflow.com/questions/1711681/how-to-best-create-a-random-float-in-a-range-between-two-float
def range (min, max)
  rand * (max-min) + min
end

def get_specific_variation(currency, price)
  selected_price = currency[price]
  new_price = range(
    selected_price*(1-MAX_VARIATION),
    selected_price*(1+MAX_VARIATION)
  )
  currency[price] = new_price
end

def zero_correction(currency, price)
  if currency[price] < 0
    currency[price] = 0
  end
end

def get_variation(currency)
  get_specific_variation(currency, "buy_price")
  get_specific_variation(currency, "sell_price")

  zero_correction(currency, "buy_price")
  zero_correction(currency, "sell_price")

  if currency.buy_price < currency.sell_price
    average = (currency.buy_price + currency.sell_price) / 2
    currency.buy_price = average
    currency.sell_price = average
  end
  return currency
end

namespace :update do
  desc "-------Update the price randomly-------"
  task :price => :environment do

    to_update_btf = Btf.first
    to_update_bth = Bth.first

    if !to_update_btf.nil? && !to_update_bth.nil?
      get_variation(to_update_btf)
      to_update_btf.save
```

Separamos las funciones

Programación defensiva

Fácil de entender

Nombres descriptivos

Lecciones Aprendidas

Setear proyecto en conjunto

Lecciones Aprendidas

Setear proyecto en conjunto

Lenguajes de programación que todo el equipo domine

Lecciones Aprendidas

No setear bibliotecas innecesarias

naranjus

Muchas gracias