Universidad Nacional de Río Cuarto
Facultad de ciencias exactas, físico-químicas y naturales.

Ingeniería de Software

Informe Cuarta Actividad

PROFESORES
Arsaute Ariel Sebastian
Daniele Marcela Elena
Solivellas Daniela Beatriz
Uva Marcelo Ariel

INTEGRANTES
Rosatti Nicolle
Tissera Joaquín

Río Cuarto - Córdoba
Noviembre 04
2024

En un primer momento, refactorizamos según los estándares de la gema Rubocop, de manera tal de:

Usar comillas simples para strings que no tienen sobrecarga.

Revisamos la longitud de los métodos que seguramente necesitarían una refactorización.

Las líneas que superan en ancho una cierta longitud, seguramente necesitarían una refactorización.

El hecho de refactorizar viene a traer beneficios a la hora de producir, mantener y leer el código. Por ejemplo, hacerlo más fácil de entender, de encontrar errores, programar más rápido.

Gracias a la herramienta Rubocop encontramos que algunos métodos eran demasiado largos en cantidad de líneas, existiendo allí un posible **code smell**.

El problema de **code smell** puede aparecer gracias a código duplicado, métodos o clases largas, mucha cantidad de parámetros, cambios divergentes (una clase cambia de diferentes formas por diferentes razones, donde en realidad dos objetos son mejor que uno), shotgun surgery (en donde es lo opuesto a cambios divergentes, un objeto es mejor que dos), y muchas más, definidas en el libro **Refactoring: Ruby Edition - Libro de Kent Beck y Martin Fowler.**

El hecho de que sean largos significaba que se podian refactorizar en módulos, de manera tal de mantener las responsabilidades por separado de las distintas cosas. Pueden tratarse como Bloaters o como Long Method / Large class.

Algunas clases refactorizadas fueron los metodos:

post '/login'
post '/register'
post '/practice'
post '/next_question'

Algunos metodos que fueron refactorizados pero que aún superaban los estándares de la gema:

def save_pdf

Algunos no podíamos refactorizarlo más de lo que estaban, de manera que agrupaban lo justo y necesario, pero no cumplía los estándares de la gema:

def generate_questions

Los nuevos métodos que encapsulan responsabilidades para los métodos refactorizados son:

def set_error_status(status_code, error_message)
def find_user_by_credentials(username_or_email, password)
def fields_missing?(params)
def find_user(username, email)
def handle_error(error_key, error_message, redirect_path)
def register_user(params)
def build_user(params)
def clean_param(param)
def clean_or_default(param, default)
def handle_successful_registration(user)
def handle_error_registration

Un ejemplo de refactorización:

Método original:

```
post "/register" do
  username = params[:username]
  name = params[:name]
  lastname = if params[:lastname].strip.empty? then "ApellidoNoRegistrado" else
params[:lastname] end
  cellphone = if params[:cellphone].strip.empty? then "CelularNoRegistrado" else
params[:cellphone] end
  email = params[:email]
  password = params[:password]
  if username.strip.empty? || name.strip.empty? || email.strip.empty? ||
password.strip.empty?
    session[:error_registration] = "missing_fields"
    logger.error "Fields Username, Name, Email and Password must be filled out. Please try
again."
    redirect "/error-register"
  else
    @user = User.find_by(username: username) || User.find_by(email: email)
    if @user
      session[:error_registration] = "user_exists"
      logger.error "An user with that username or email already exists. Please try a different
one."
      redirect "/error-register"
    else
      isAdmin = 0
      @user = User.create(username: username, name: name, lastname: lastname,
cellphone: cellphone, email: email,
                          password: password, isAdmin: isAdmin)
      if @user.persisted?
        session[:isAnUserPresent] = true
        session.delete(:error_registration)
        session[:user_id] = @user.id
        redirect "/"
      else
        session[:error_registration] = "user_not_persisted"
        logger.error "There was a problem registering your account. Please try again later."
        redirect "/error-register"
      end
    end
  end
end
```

Método refactorizado:

```
post '/register' do
  if fields_missing?(params)
    handle_error('missing_fields', 'Fields Username, Name, Email and Password must be
filled out. Please try again.',
                 '/error-register')
  else
    user = find_user(params[:username], params[:email])
    if user
      handle_error('user_exists', 'An user with that username or email already exists. Please
try a different one.',
                   '/error-register')
    else
      register_user(params)
    end
  end
end
```

tal que los métodos nuevos son quienes resuelven el método en sí, ayudando a la modularización y distribución de responsabilidades.

En una segunda refactorizacion, según refactorización por MVC, desglosamos responsabilidades que se reparten entre controladores, vistas y servicios.

Anteriormente, el archivo app.rb era responsable de mantener las rutas y las definiciones de las cosas.

Actualmente, luego de la refactorización, tenemos responsabilidades según:

Usuario:
- user_controller.rb
- user_services.rb

Documentos:
- documents_controller.rb
- document_services.rb

Aprendizaje del usuario / Prácticas:
- practice_controller.rb
- practice_services.rb

Controladores de propósito general:
- main_controller.rb

Servicios de propósito general:
- utils.rb

Ahora las responsabilidades pueden mantenerse más fácil, el código es más legible, se puede encontrar errores de manera ágil y otros beneficios a futuro de contener la modularización y refactorización de los métodos.