

Actividad 4 Taller

Refactorización

La **refactorización en Ruby** es una práctica esencial que busca mejorar la estructura interna del código sin alterar su comportamiento externo. Esta técnica se enfoca en hacer el código más legible, mantenible y eficiente, simplificando su lógica y eliminando redundancias. Refactorizar permite identificar y corregir posibles errores y mejora la capacidad del código para adaptarse a futuros cambios, haciendo que el desarrollo sea más ágil y de mayor calidad.

Integrantes

Conti Bruno
Gonzalez Juan Cruz
Mezzano Joaquin
Vollenweider Erich

5 de noviembre de 2024

Una de las primeras cosas que hicimos fue instalar la gema Rubocop. Una vez instalada la gema, al ejecutarla por primera vez, con el comando “rubocop” el resultado fue el siguiente:

```
31 files inspected, 2775 offenses detected, 2652 offenses autocorrectable
```

Sabiendo esto, y según la información provista por la página oficial de la gema, ejecutando “rubocop -a”, esas 2652 ofensas se deberían solucionar automáticamente sin modificar y sin romper nada de la app.

El siguiente paso fue ejecutar el comando “rubocop -a”, con lo que obtuvimos:

```
31 files inspected, 5879 offenses detected, 5710 offenses corrected, 45 more offenses can be corrected with `rubocop -A`
```

Según la información provista por la gema, todavía podemos seguir corrigiendo ofensas de manera automática con el comando “rubocop -A”. Luego de ejecutar ese comando, el resultado fue:

```
31 files inspected, 203 offenses detected, 76 offenses corrected
```

Para chequear que no quedaran ofensas que se pudieran resolver de manera automática, volvimos a ejecutar el comando “rubocop”, el cual nos dio la siguiente información:

```
31 files inspected, 127 offenses detected
```

Seguimos corrigiendo ofensas más que nada para cumplir con alguna de las reglas de estilo reportadas por Rubocop, obteniendo lo siguiente:

```
31 files inspected, 100 offenses detected
```

Unos de los errores que Rubocop detectó es que había 12 archivos que no tenían comentarios que explicaran cuál era el objetivo de los mismos. Una vez resuelto ese problema los resultados fueron los siguientes:

```
31 files inspected, 87 offenses detected
```

También detectó un error el cual era una variable global llamada racha. Para solucionar este problema decidimos crear una columna llamada racha en la tabla users. Esta solución nos pareció la más adecuada ya que el usuario no va a perder la racha acumulada si desea cerrar la aplicación. Estos fueron los resultados obtenidos:

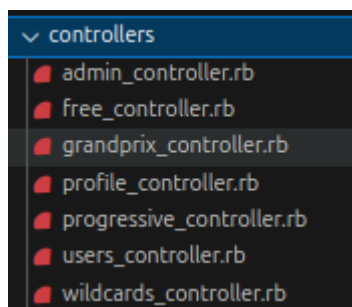
```
31 files inspected, 66 offenses detected
```

Luego de esto empezamos a definir métodos, en base a métricas methodLength-blockLength y ABCsize. Además de separarnos en ramas para que cada uno pudiera resolver la actividad que le había tocado.

Detectamos que la clase 'app.rb' era muy extensa, gracias al reporte de Rubocop, por lo que decidimos separar esa clase en controladores más pequeños que encapsularon métodos relacionados entre sí. De esta manera vamos a lograr mejorar la cohesión y asegurar que cada controlador solo sea responsable de una tarea, facilitando futuras modificaciones y el mantenimiento del código.

Controladores creados:

- users_controller
- admin_controller
- profile_controller
- wildcards_controller
- progressive_controller
- grandprix_controller
- free_controller



y la app usa todos los controladores

```
require_relative './controllers/users_controller'
require_relative './controllers/profile_controller'
require_relative './controllers/wildcards_controller'
require_relative './controllers/admin_controller'
require_relative './controllers/grandprix_controller'
require_relative './controllers/free_controller'
require_relative './controllers/progressive_controller'

set :database_file, './config/database.yml'
set :public_folder, "#{File.dirname(__FILE__)}/public"

enable :sessions

# Esta clase define la aplicación principal utilizando Sinatra.
# Controla las rutas y la lógica de la aplicación web.
class App < Sinatra::Application
  use UsersController
  use ProfileController
  use WildCardsController
  use AdminController
  use GrandprixController
  use FreeController
  use ProgressiveController
end
```

A medida que fuimos incorporando estos controladores pudimos ver como las ofensas que nos indicaba rubocop iban bajando, además de que el código quedaba más entendible y más fácil de corregir aquellos errores que íbamos encontrando.

```
35 files inspected, 65 offenses detected
```

Además para modularizar aún más el código, creamos los llamados “módulos” en ruby, en los cuales guardamos métodos que habíamos utilizado en otros archivos, como por ejemplo en los controladores, con el objetivo de simplificar aún más el código y que sea mucho más legible.

Módulos creados:

- free_logic
- grandprix_logic
- progressive_logic

Con esa implementación obtuvimos el siguiente avance:

```
41 files inspected, 47 offenses detected
```

Corregimos más ofensas de los controladores , creando una carpeta de helpers que contiene el archivo helpers.rb el cual contiene todas las sesiones activas.

Luego creamos un archivo de configuración para rubocop llamado .rubocop.yml donde tomamos en cuenta la idea de que el schema.rb (archivo de la estructura de la base de datos, que se crea automáticamente cuando se ejecuta la aplicación) no tendría que ser una ofensa , y además agregamos que la métrica de methodLength sea de 15 en cambio de 10 ya que teníamos varios métodos donde las líneas eran 11/12 y se podrían haber modificado pero quedaba más engorroso de lo que ayudaba a la refactorización, por eso decidimos hacer este cambio.

```
AllCops:
  Exclude:
    - 'db/schema.rb'
  TargetRubyVersion: 3.1 # para que no tenga problema con el versionado
  NewCops: disable

Metrics/MethodLength:
  Max: 15
```

archivo .rubocop.yml

Además seguimos modularizado y achicando tamaño de la misma línea (máximo 100 caracteres), de métricas como blockLength y methodLength quedando un total de 47 ofensas.

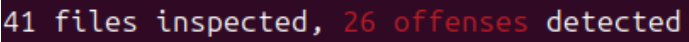
Luego corregimos los problemas ABCsize (complejidad) también con la idea de módulos nuevos , los cuales nos deja con 41 ofensas:

```
41 files inspected, 41 offenses detected
```

Quedamos actualmente con 41 ofensas que son parte de la carpeta spec (test) con

métodos y bloques de longitud.

Luego de retocar un poco los archivos de test de las creaciones y dividir un describe en varios describes las ofensas que tenemos son 26 aun.



41 files inspected, 26 offenses detected

Conclusiones: Dada esta gema rubocop pudimos refactorizar es decir no modificar el comportamiento pero si modularizar, cambiar, organizar mejor nuestro código con distintas métricas como por ejemplo cantidad de los bloque de módulos o la complejidad de los mismos, donde antes teníamos básicamente todos los endpoints y métodos en el app.rb ahora actualmente tenemos diferentes carpetas y cada una juega un rol importante de responsabilidad , que además esto nos aporta muchas más claridad a la implementación y por lo tanto al sistema en sí.

Al refactorizar independientemente del lenguaje en sí tenemos muchas ventajas como por ejemplo de las pruebas, más que nada que si se rompe algo sabemos donde tocar o donde ir, luego otras ventajas obviamente podrían ser el mantenimiento o la extensionalidad que ahora es más fácil la extensión si se quisiera de nuevas funcionalidades.