

ACTIVIDAD N°4 - Taller IS 2023.

- Leer la bibliografía sugerida “Refactoring: Ruby Edition - Libro de Kent Beck y Martin Fowler.
- Instalar y ejecutar la herramienta Rubocop.
- Analizar los reportes generados por la herramienta Rubocop.

Se instaló la herramienta RUBOCOP. Una vez instalada, se ejecutó `$ rubocop -o informeApp`, y el resultado fue el siguiente:

El output del resultado generado se encuentra en `cheftravel/docs/informeApp`

```
03                                     ^^^
04 spec/spec_helper.rb:26:4: C: [Correctable] Layout/TrailingEmptyLines: Final newline m
05 end
06
07
08 37 files inspected, 283 offenses detected, 212 offenses autocorrectable
09 |
```

Una vez analizadas las ofensas, ejecutamos el comando para corregir automáticamente las infracciones: `$ rubocop --auto-gen-config`.

El resultado fue el siguiente:

```
spec/spec_helper.rb:14:39: C: [Corrected] Style/RedundantFileExtensionInRequire: Redundant .rb file extension detected
require_relative '../config/enviroment.rb'
                                ^^^
spec/spec_helper.rb:26:4: C: [Corrected] Layout/TrailingEmptyLines: Final newline missing.
end
```

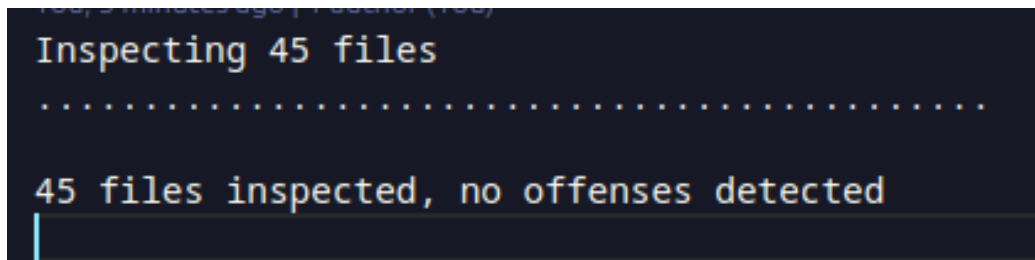
Volvimos a correr Rubocop:

```
210 spec/models/server_spec.rb:8:1: C: Metrics/BlockLength: Block has
211 RSpec.describe 'Sinatra App' do ...
212 ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
213 spec/models/user_spec.rb:7:1: C: Metrics/BlockLength: Block has t
214 describe User do ...
215 ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
216 spec/spec_helper.rb:13:121: C: Layout/LineLength: Line is too lon
217 ActiveRecord::Base.establish_connection(adapter: 'sqlite3', datab
218 |
219
220 37 files inspected, 71 offenses detected
221 |
```

- Ejecutar un ciclo de refactorización de su proyecto. Es muy importante tener en cuenta los siguientes pasos:
  - Asegurarse que los test estén completos antes de alterar el código. Dichos test deben pasar antes y después de los cambios realizados.
  - Todo cambio que se realice debe estar justificado mediante alguna de las reglas de refactorización estudiadas o bien para cumplir con alguna de las reglas de estilo reportadas por Rubocop.

Antes de alterar el código, completamos y corregimos los tests, ya que al correrlos, varios de ellos fallaban, esto se debió a las últimas funcionalidades agregadas en el sprint anterior, como lo fué la encriptación de la contraseña del usuario y un arreglo que se llevó a cabo en el tratamiento de puntos y niveles del usuario. Se continuó trabajando con PivotalTracker y con la metodología de branches en git. Los errores corregidos en los tests son el resultado de la historia de usuario 'complete tests'.

Hemos llevado a cabo un ciclo de refactorización en nuestro proyecto, identificando y abordando varios "Code Smells". Los principales problemas que enfrentamos fueron métodos largos (Long Method) y clases muy grandes (Large class) en el archivo server.rb. Nuestra estrategia de refactorización se basó en asegurarnos de que los tests estuvieran completos antes de realizar cualquier cambio, y garantizar que estos tests pasaran tanto antes como después de los cambios. Además, justificamos cada cambio mediante reglas de refactorización estudiadas o reglas de estilo reportadas por Rubocop.



```
Inspecting 45 files
.....
45 files inspected, no offenses detected
```

### Pasos Seguidos:

#### 1) Aseguramiento de Pruebas Completas:

Antes de realizar cualquier cambio, nos aseguramos de que nuestras pruebas estuvieran completas y abarcaran todas las funcionalidades afectadas por las modificaciones planeadas. Esto garantizó que cualquier cambio no rompiera la funcionalidad existente.

#### 2) Identificación de Code Smells:

Durante la revisión del código, identificamos varios problemas de calidad de código en nuestro archivo server.rb. Los principales Code Smells fueron métodos largos y clases grandes, lo que dificulta la legibilidad y el mantenimiento del código.

#### 3) Refactorización de Métodos Largos:

Abordamos el problema de los métodos largos (Long Method) descomponiendo las funciones largas en partes más pequeñas y legibles. Cada función se centró en una única responsabilidad, lo que mejoró la modularidad y la claridad del código. Esto también permitió la reutilización de código.

#### 4) Refactorización de Clases Grandes:

Para abordar el problema de las clases grandes (Large class), dividimos la lógica en varias clases más pequeñas, cada una responsable de una parte específica de la funcionalidad. Esto mejoró la cohesión y redujo la complejidad del código.

#### 5) Eliminación de Duplicaciones:

Donde encontramos duplicación de código, aplicamos el principio "Don't Repeat Yourself" (DRY) para eliminar repeticiones innecesarias y promover la reutilización de código.

#### 6) Validación con Reglas de Estilo (Rubocop):

Utilizamos Rubocop para asegurarnos de que nuestro código cumpliera con las reglas de estilo. Cualquier violación de las reglas de estilo se abordó según las recomendaciones de Rubocop.

#### Revisión y Pruebas:

Después de realizar los cambios, revisamos cuidadosamente el código y volvimos a ejecutar todas las pruebas. Nos aseguramos de que tanto las pruebas anteriores como las nuevas pasaran sin problemas.

```
05:53:15 agustin@agustin-HP cheftravel ±|186328463-refactor-project X|→ sudo docker  
.....  
Finished in 5.14 seconds (files took 1.32 seconds to load)  
25 examples, 0 failures
```

#### Resultado:

Como resultado de este ciclo de refactorización, nuestro código se ha vuelto más claro, modular y fácil de mantener.

#### Recomendaciones:

Continuaremos vigilando la calidad del código y realizando ciclos de refactorización periódicos durante el transcurso de la materia para abordar cualquier nuevo Code Smell que surja. También es importante mantener nuestras pruebas actualizadas y completas para garantizar que cualquier cambio futuro se pueda validar de manera efectiva.