Informe Actividad N° 4 - Taller Ingeniería de Software

Integrantes de grupo: Fernandez Manuel, Moreno Michael, Urquiza Carlos

Los objetivos propuestos para esta actividad fueron:

● 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 Rubocop.

● 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.

● Producir un informe en donde se reporte el trabajo realizado. Este informe deberá presentar el o los problemas encontrados por la herramienta, o bien detectados manualmente junto las acciones ejecutadas para corregir dichos problemas. Cada modificación deberá estar justificada y fundamentada en los code smells detectados y las reglas de refactorización aplicadas (analizadas en la bibliografía).

Informe generado a partir de la ejecucion de la herramienta Rubocop

Luego de una previa etapa de refactorización en el proyecto, se dedicó un sprint completo a completar la refactorización de métodos del server y otros CodeSmells detectados.
Se ejecutó la herramienta RuboCop por primera vez en todo el proyecto y se detectaron 567 ofrendas, de las cuales solo 518 fueron autocorregibles. Luego se ejecutó el comando `rubocop --auto-correct` y pasamos a tener 85 ofensas, y 42 autocorregibles. Teniendo esto, se ejecutó rubocop -a y pasamos a tener 123 ofensas y 80 autocorregibles.
Cuando corrimos por 2 vez `rubocop --auto-correct` conseguimos 43 ofensas.
Ese fue el valor más bajo obtenido ejecutando la herramienta.
A partir de aquí, la herramienta RuboCop señala lo siguiente:

- La herramienta sugiere documentación al comienzo de cada clase y modelo.
- En la migración de user, sugiere acortar el tamaño de la definición (12 líneas de longitud contra 10 líneas de longitud que sugiere Rubocop)
- Sugiere un acortamiento del esquema (imposible)
- La clase User es demasiado grande (115 líneas contra 100 que sugiere)
- Sugiere la refactorización del método update_role, por su longitud
- La refactorización completa de buy_card por: longitud de método, alta complejidad ciclomática [16/7] y por uso de una condición muy grande.

- Refactorizar el server, ya que es demasiado grande y contiene muchas operaciones que podrían pasarse a los modelos (server tiene 236 líneas, cuando lo óptimo sería 100).
- Post de responses (relacionado a la respuesta de un usuario) demasiado largo (41/25)
- Por último, sugiere un acortamiento en los bloques que se usan para realizar test de las clases: Progress, app, question y User

Los principales CodeSmells que detectó el equipo en base a Rubocop y la lectura de la bibliografía recomendada fueron: Clases largas, métodos largos y repetición de código
A partir de las ofensas marcadas por Rubocop, se propuso el siguiente orden de tareas para tratar los codeSmells que se detectaron.Se arreglaron los test que se encontraron en falla luego de las modificaciones al código, dejando una suite de test con el coverage más alto posible y sin errores.
Se refactorizaron principalmente métodos get y post del server de la aplicación, por mencionar algunos de ellos, el post de response, el post de user, y además el get de progress.
La clase User fue la clase con mayor cantidad de CodeSmells detectados, en ella se encontraron métodos demasiado largos, repetición de algunos fragmentos de código y además y en general la clase era demasiado grande, superando las 100 líneas de código.
Para solucionar el código repetido se creó una lista con la información para luego acceder a ella en los distintos métodos y no repetirlo. Con esto también se redujo la cantidad de líneas que tienen los métodos que utilizaban esa lista y en consecuencia, la clase también redujo su tamaño.
Al final del sprint, el equipo agregó los controladores para los diferentes métodos post y get que había en el Server para el usuario, su progreso, la tienda y otras funcionalidades de la aplicación.
Como resultado quedaron 4 archivos de controladores:

- user_controller
- game_controller
- others_controller
- store_controller

De esta forma, el server de la aplicación quedó más reducido y cada controlador puede ampliarse y seguirse dividiendo si es necesario.