SUG-MATH

INFORME DE REFACTIRIZACIÓN

Proyecto: SUG-MATH

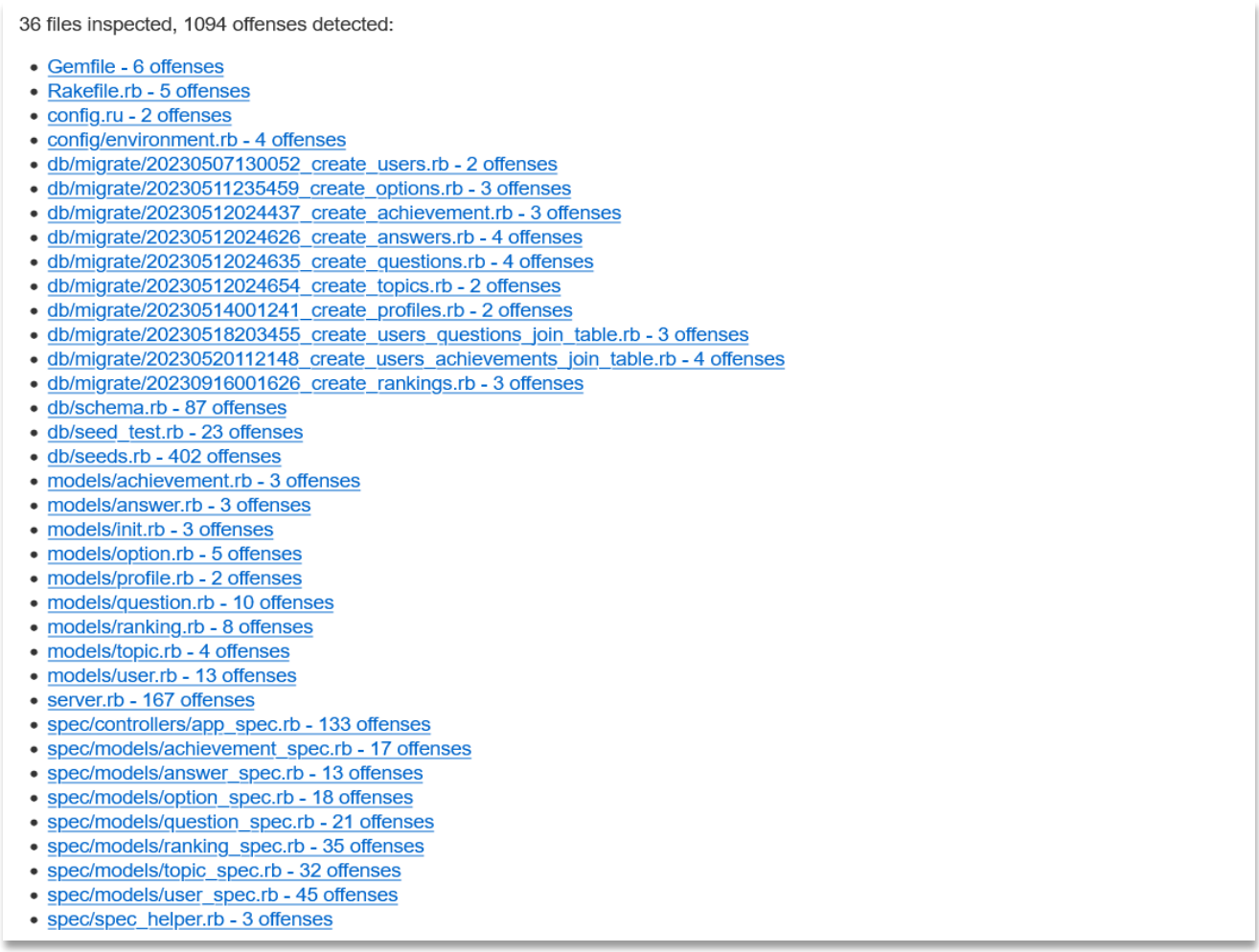
Estudiantes: Buchieri Giovanni
Guzman Alexis Uriel
Vázquez Santiago

Actividad 4: Refactorización de código

Ante la falta de experiencia presentada por el equipo en el transcurso del desarrollo de aplicación en cuestión, el diseño, el modelado y la organización de clases, como asi también de componentes de la aplicación se vieron afectados de manera negativa. Como problema principal que se afrontara en el siguiente informe, se tiene la falta de un apartado de controladores, y una mala distribución de responsabilidades en cuanto al diseño del controlador principal Server.rb, por ende, nos abocaremos en refactorización de este aspecto para una mejor legibilidad, mayor comprensión por parte del lector y la incorporación/corrección de ofensas en los estándares de diseño, propias del lenguaje Ruby.

Primera vista del proyecto:

Como se puede apreciar en la siguiente captura de la primera ejecución de la gema rubocop, se inspeccionaron 36 archivos con extensiones de Ruby, entre ellos como se ve en la captura extraída de la terminal, se encontraron 1094 ofensas a los estándares previamente mencionados, de los cuales 1009 son auto corregibles por la herramienta. Sin dejar de lado el gran numero, vale aclarar que la mayoría de los posibles cambios que son detectados, se encuentran en el archivo Seed.rb, es decir, el diseño principal tiene que sufrir mejoras, pero no representa el peor caso en el que se podría encontrar.



1.1 Primera vista del diseño presente en la aplicación

```
36 files inspected, 1094 offenses detected, 1009 offenses autocorrectable
```

1.2 Numero inicial de correcciones a realizar en términos de sintaxis

Separación de controladores:

Como primera acción de refactoreo, se dividió el controlador principal (archivo server.rb) en sub controladores, que contienen un subconjunto de las rutas del mismo. Esto ayuda a la distinción de diferentes funciones de la ampliación, como para el testeo de las mismas. El porqué de este cambio es claro, *Code Smells: Large Class and Divergent Changes*, todo el código estaba contenido en un solo archivo, cada vez que surgía una idea de cambio se modificaba ahí, con lo que la mantención de este era extremadamente costosa, tomaba tiempo encontrar el método o sección que se quería trabajar, etc. A continuación, se presentan el cambio realizado con su ya mencionada división.

1.3 Rediseño de rutas en términos de controladores

Además, para facilitar posteriores correcciones, estos controladores se escribieron de la manera mas correcta posible, intentando omitir todo tipo de imperfección en las convenciones y en la sintaxis del lenguaje Ruby. Este trabajo elimino los siguientes *Code Smells*: *Dupliacate Code, Long Method and Unnecessary Comments*. Dichos imperfectos se corrigieron utilizando los métodos de refactorización mencionados a continuación: *Extract Variable, Extract Method and Previous Refactor.*

Re ejecución de la herramienta rubocop:

En este punto tras haber hecho una división y corrección tan exhaustiva sobre el principal problema que presentaba el sistema, la gran clase App, se volvió a ejecutar Rubocop para verificar que las modificaciones impuestas, efectivamente dieron solución a una parte de las ofensas encontradas en el primer análisis, lo que resulto en un indudable éxito, dado que esta pequeña acción redujo el primer valor obtenido en aproximadamente 121 ofensas.

```
42 files inspected, 971 offenses detected, 887 offenses autocorrectable
```

1.4 Numero resultante de correcciones a realizar tras división de controladores

Ejecución de la herramienta de auto reparado de Rubocop:

Para agilizar la búsqueda y solución de ofensas, ejecutamos la extensión de Rubocop que nos permite la auto corrección de las mismas, o por lo menos aquellas que están a su alcance de programa. Una vez realizada esta acción los resultados fueron favorables, frente al número visto previamente se llegó a tener aproximadamente 125 ofensas menos, un valor mejor al anterior pero no demasiado bueno aún. Esto sucedió ya que se ha llevado a cabo únicamente las autocorrecciones que el sistema considera como “seguras”, es decir, aquellas que no modifican en absoluto el funcionamiento o la estructura de las clases, de los métodos, o demás secciones del programa donde se encuentren.

```
42 files inspected, 846 offenses detected, 761 offenses autocorrectable
```

1.5 Numero resultante de correcciones tras “autocorrección”

Configuración y re ejecución del auto reparado:

Tras un análisis minucioso de donde eran que estaban las ofensas, y cuales eran, notamos que habían muchas trivialidades, tales como que cada clase y cada método debían tener una "documentación" propia, es decir, una sección de comentarios previa al mismo, o que por convenciones las variables de mas de una palabra debía estar escritas con notación snake_case, sin dudas, una serie de reglas que para nuestro propósito no eran indispensables, y en algunos casos, ni siquiera eran útiles. Asique, luego de configurar un archivo rubocop_todo.yml con lo que podríamos decir, dejamos pasar, vimos que el número de ofensas había bajado drásticamente un número aproximado de 122.

```
42 files inspected, 734 offenses detected, 719 offenses autocorrectable
```

1.6 Numero resultante de correcciones tras omitir las innecesarias

Para finalizar con la parte de autocorrección, usamos nuevamente la herramienta, pero esta vez en modo de corrección completa, de esta manera la mayoría de las ofensas de Style, Format, Layout, etc., se corrigen entre los 42 archivos. Con este paso se solucionan alrededor de 711 ofensas, dejándonos solo aquellos *Code Smells* que deberíamos corregir manualmente por la razón mencionada previamente de que el programa no encontró una solución para ellos, sea por peligro a modificación inapropiada de alguna línea o simplemente por no saber cómo.

```
42 files inspected, 33 offenses detected
```

1.7 Numero resultante de correcciones autocorrección final

Reparaciones restantes de archivos con ofensas:

Como ultimo paso de esta reparación, analizamos detenidamente los problemas que faltaban y la mayoría eran exceso en longitud de líneas, *Large Class* testeo de rutas en el archivo app_spec y un cambio de nombre en el archivo Rakefile, el cual debía estar con minúscula o con sintaxis snake_case. Dados que eran eso los puntos faltantes, como primera acción agregamos al rubocop_todo.yml la omisión del largo de línea, ya que esto nos lo complicaba el archivo seed, el cual tiene algunas líneas de preguntas que son bastantes extensas. Y como ultimo arreglo hicimos lo mismo desde la perspectiva de los test de ruta, esto porque no era mucho lo que faltaba para que se cumpliera la media solicitada y porque sacamos un par de *Dead Lines* que estaban comentadas. Una vez realizadas las acciones anteriores, conseguimos una cobertura completa de la herramienta como se ve a continuación.

```
Inspecting 42 files
..........................................

42 files inspected, no offenses detected
```

1.8 Resultado de toda la corrección realizada

Reparaciones de cambios de modelos:

Tras solucionar todo lo relacionado con estilo, se realizaron cambios en archivos con la lógica para mejorar el diseño del mismos. En la siguiente imagen se ve que solo se encontraron 15 ofensas, todas solucionables internamente por rubocop, esto sucede ya que la mayoría son cuestiones de estilo que decidimos en algunos pasos previos omitir.

```
42 files inspected, 15 offenses detected, 15 offenses autocorrectable
```

1.9 Resultado de cambios en modelos

Por la razón mencionada usamos rubocop una última vez para que quede lo más uniforme posible el resultado y porque eran detalles mínimos, asi se solucionaron todos los problemas dejando el análisis en la misma condición que la imagen 1.8, mostrada arriba.

Corrección de test:

Como era de esperarse, luego de haber añadido funcionalidades en iteraciones pasadas, luego de reacomodar y refactorizar el controlador principal, el archivo server, surgieron fallos en la ejecución de los test. En su mayoría no daban fallos, sin embargo, había un subconjunto de pruebas que trabajaban sobre las rutas que anteriormente estaban en server, con problemas.

Al realizar algunos cambios necesarios y ejecutar los test, nos encontramos con un problema adicional. Al ejecutar el comando necesario para ejecutarlos, no se reconocía la base de datos específicamente creada para las pruebas. Además, durante este proceso, identificamos errores en los archivos de configuración de las bases de datos que estaban afectando el funcionamiento de los test.

```
An error occurred while loading spec_helper.rb.
Failure/Error: require File.expand_path('../config/environment.rb', __dir__)

Psych::BadAlias:
  Unknown alias: default
ook 3.47 seconds to load)
0 examples, 0 failures, 1 error occurred outside of examples
```

1.10 Errores en ejecución de test

Para abordar esta situación, procedimos a corregir los errores en los archivos de configuración. Luego, ejecutamos los test uno por uno, realizando las correcciones necesarias, hasta que finalmente pasaron todos de manera satisfactoria.

```
.................................

Finished in 21.13 seconds (files took 4.87 seconds to load)
53 examples, 0 failures
```

1.11 Configuración Corregida de test

Terminando con una cobertura:

All Files (97.45% covered at 1.48 hits/line)

25 files in total.
550 relevant lines, **536** lines covered and **14** lines missed. (97.45%)

1.12 Cobertura de los test

Conclusiones:

El trabajo de refactorización si bien es un proceso que lleva tiempo, cuando el código no se escribió desde un principio con una estructura sólida y en base una idea bien definida, se convierte en problema, por ende, al estar diseñando cualquier proyecto de software, este se debería realizar de la manera más consciente, ordenada y metódica posible. Si bien este no fue el caso en que se tuvo que realizar una remodelación completa del programa, de sus directorios, base de datos, entre otras cuestiones, de igual manera llevo un tiempo saber cómo y dónde acomodar lo que se tenía para respetar el esquema general de los proyectos en Ruby.