**Universidad Nacional de Río Cuarto**

**Facultad de Cs. Exactas Físico Químicas y Naturales**

**Departamento de Computación**

**Ingeniería de Software (Cód. 3304) - 2023**

# Taller IS 2023 - Actividad Nro: 4

**Integrantes:** Darío Ferreyra y Kevin Riberi

**Refactorización**

Agregamos la gema Rubocop con la siguiente línea en el archivo Gemfile:

```
gem 'rubocop', '~> 0.91.0', :require => false
```

En el cuatrimestre pasado se había comenzado una refactorización de los post y get del server.rb a la carpeta controllers. La primera modularización fue:

https://github.com/kevinriberi/AYDS-Taller/commit/c96a4b991279af6af0a51966fe0fa6023f49fb7f#diff-6eb2df8dc0ebd6ce27bd4f668a9337504ab6c1f648dd1c81d3209bf5647ad4b6

Luego se fueron realizando pequeños cambios al server.rb hasta lograr un desacoplamiento óptimo al trasladar la lógica de enrutamiento a la carpeta controllers, lo que hizo que el archivo server.rb sea más limpio y tenga una responsabilidad más clara en la configuración del servidor web.

También se refactorizó el archivo user_controller.rb agregando los siguientes métodos:

- username_taken?(email)
- email_taken?(email)
- passwords _match?(password, confirm_password)
- create_user(username, email, password)

Este es el link donde se muestra ese estado del user_controller.rb:

https://github.com/kevinriberi/AYDS-Taller/commit/381f24c26e4396dbf4ff1fff12f6c223409ebdc6#diff-140bfa288c2b8181905ce2251aaa2a2f2304be5dab04406601eb5446d4c49b09

Luego se aplicó la técnica “Move method” para mover estos métodos al modelo user.rb. El estado del proyecto en esta instancia es la siguiente:

https://github.com/kevinriberi/AYDS-Taller/commit/65a604bb419ccf99538a4f6015dce3d421e22925#diff-140bfa288c2b8181905ce2251aaa2a2f2304be5dab04406601eb5446d4c49b09

Luego se realizó una reorganización de enrutamiento por funcionalidades, por lo que se tuvo que agregar y eliminar clases de controladores. Este es el estado temporal de controllers en ese momento:

https://github.com/kevinriberi/AYDS-Taller/commit/29e6cfd975c441e715093626a9f078678392d487

Finalmente, despues de otros cambios pequeños adicionales, cambios de nombres y reordenamiento de rutas, se obtuvo el siguiente estado de controllers:

https://github.com/kevinriberi/AYDS-Taller/commit/a294deda0f545aea2b3889eef83c39c90a3a95e4

Antes de ejecutar la herramienta rubocop nos aseguramos de que los test estén completos antes de alterar el código. Agregamos tests para algunas rutas que faltaban. Se pueden observar los cambios en este link:

https://github.com/kevinriberi/AYDS-Taller/commit/618a0d751a323208015da5c8a5fdeee8340b55ec

Y finalmente se refactorizaron los test. Se utilizó el after y before para la creación y destrucción de datos:

https://github.com/kevinriberi/AYDS-Taller/commit/5b49d5909d2efca50f25fe377ab4f5fc945ed519

Una vez que se establecieron todas las posibles refactorizaciones, ejecutamos la herramienta rubocop para que analizará nuestro código ruby buscando posibles ofensas que hayamos cometido, y luego evaluar las sugerencias de cómo resolverlas para que nuestro código sea más óptimo y mantenible.  Para ello se utilizó el siguiente comando:

```
$ rubocop
```

Los primeros resultados de la salida de la consola se guardaron en todos los archivos que se encuentran en AYDS-Taller/doc/rubocop. Los nombres de los archivos.txt se corresponden a los comandos ejecutados y están enumerados en el orden de ejecución.

En la primera ejecución se detectaron 1140 ofensas en 56 archivos, de los cuales todos eran autocorregibles (archivo 1-0_rubocop.txt). Se ejecutó de nuevo $ rubocop para ver si mantenía el mismo resultado que antes, pero se detectaron 950 ofensas en la misma cantidad de archivos (archivo 1-1_rubocop.txt). Luego se ejecutó:

```
$ rubocop --auto-gen-config
```

Y se generó un archivo un archivo de configuración ”rubocop_todo.yml” el cual se configuró en base a lo que podía modificar automáticamente y lo demás lo ignora. Luego se ejecutaron los comandos para realizar los comandos:

```
$ rubocop -a
```

```
$ rubocop -A
```

rubocop -a realiza correcciones menores de estilo y convención, mientras que rubocop -A realiza correcciones más amplias que pueden afectar la estructura del código o abordar problemas de seguridad y rendimiento. Luego se realizó una vez más
$ rubocop -A

y

$ rubocop

y no se detectó ninguna ofensa más para corregir.

Un problema no detectado por rubocop se encontró en el modelo knowledge y se solucionó con la técnica de refactorización Inline Temp:

Código original:

```
   def percentage_of_correct_answers

     if self.is_finished

       return 100

     end

     level_up_threshold =
topic.send("amount_questions_L#{level}".to_sym)

     rate = self.correct_answers_count.to_f / level_up_threshold

     percentage = (rate * 100).round(1) # Redondear a 1 decimales

     return percentage

   end
```

Código luego de aplicar la regla

```
      def percentage_of_correct_answers

     if self.is_finished

       return 100

     end

     rate = self.correct_answers_count.to_f /
topic.send("amount_questions_L#{level}".to_sym)

     percentage = (rate * 100).round(1) # Redondear a 1 decimales

     return percentage

   end
```

Dentro del archivo .rubocop_todo.yml generado por el comando $ rubocop --auto-gen-config se comentó la linea 24 que es ‘models/knowledge.rb’:

```
Layout/AccessModifierIndentation:

 Exclude:

   - 'models/answer.rb'

   # - 'models/knowledge.rb'

   - 'models/option.rb'

   - 'models/question.rb'

   - 'models/topic.rb'
```

Con esto se logra que ese archivo no sea excluido de la inspección que realiza rubocop.
Luego se ejecuto:
$ rubocop --format html --out reports/html/index.html

y se obtuvo una ofensa, la cual indicaba a la forma de poner en privado un par de métodos.
El código original era:

```
  private

  def valid_user_id

    return if User.exists?(user_id)

     errors.add(:user_id, "does not exist")

  end

   def valid_topic_id

    return if Topic.exists?(topic_id)

     errors.add(:topic_id, "does not exist")

  end
```

Luego de solucionar el problema:

```ruby
def valid_user_id

  return if User.exists?(user_id)

   errors.add(:user_id, "does not exist")

end

private :valid_user_id

 def valid_topic_id

  return if Topic.exists?(topic_id)

   errors.add(:topic_id, "does not exist")

end

private :valid_topic_id
```

Los informes de estos estados se encuentran en la carpeta /doc/rubocop HTML/ donde los archivos index enumerados 1, 2 y 3 son el estado antes de comentar la línea ‘models/knowledge.rb’ en .rubocop_todo.yml, el estado luego de comentarla y finalmente después de solucionar el problema.