TALLER DE INGENIERÍA DE SOFTWARE

ACTIVIDAD 4

INFORME DE TRABAJO

GALUCCI ESTEBAN, HERNANDEZ FLORENCIA, TREPIN GONZALO

Introducción

En la refactorización, utilizamos la gema Rubocop de Ruby para mejorar el código, de acuerdo a las buenas prácticas del lenguaje, para que sea más limpio y legible, mejorar la eficiencia del programa y prevenir errores comunes, sin cambiar su funcionamiento externo.

Correcciones realizadas en el programa

1. Comentarios

Siguiendo lo que dice la bibliografía, eliminamos muchos comentarios innecesarios que aclaraban que hacían métodos, los cuales eran muy transparentes de acuerdo a su funcionamiento. Y en los que no era muy transparente, los dividimos en dos o más métodos, haciendo cada uno más concreto y transparente.

Además agregamos comentarios que solicitaba agregar la gema, por ejemplo en modelos, describiendo el nombre del modelo y el comentario inicial: `# frozen_string_literal: true` el cual se utiliza para mejorar el rendimiento y reducir el uso de memoria al trabajar con cadenas de texto. A continuación incluímos un ejemplo de los comentarios modificados en el modelo progressLesson.rb. Antes y después de la corrección:

```ruby
class ProgressLesson < ActiveRecord::Base
    belongs_to :user
    belongs_to :lesson
    belongs_to :level
end
```

```ruby
# frozen_string_literal: true

# ProgressLesson model
class ProgressLesson < ActiveRecord::Base
  belongs_to :user
  belongs_to :lesson
  belongs_to :level
end
```

2. Métodos muy largos

   Teníamos situaciones como la siguiente donde algunos métodos y endpoints eran extremadamente largos lo cual no es deseable ya que nos trae muchas desventajas, como dificultad para la comprensión del código, pruebas más complicadas, mayor riesgo de errores, etc.

   Este es un caso que teniamos donde el endpoint /admin/questions era extremadamente largo

```ruby
post '/admin/questions' do
    halt 403, 'Access denied.' unless current_user.admin?

    question_text = params['question']
    correct_answer_text = params['correct_answer']
    question_level = params['question_level']
    topic_id = params['topic_id']

    redirect '/admin/questions/new' if topic_id.nil?

    lesson = Lesson.find_by(topic_id: topic_id)
    level = Level.find_by(lesson_id: lesson.id, number: question_level)
    imagepath = nil

    if level.nil?
      @topics = Topic.all
      return erb :new_question, locals: { error: 'Failed to create question.' }
    end

    exam = Exam.find_by(lesson_id: lesson.id, level: level.id)

    question = Question.new(question: question_text, topic_id: topic_id)

    puts params['image']

    if Topic.find(topic_id).topic == 'Banderas' && params['image'][:filename].end_with?('.svg')
      filename = params['image'][:filename]
      file = params['image'][:tempfile]

      image_folder = './public/images/flags'

      filepath = "#{image_folder}/#{filename}"
      File.open(filepath, 'wb') do |f|
        f.write(file.read)
      end

      imagepath = "images/flags/#{filename}"
    end

    if question.save
      correct_option = Option.create(response: correct_answer_text, topics_id: topic_id)

      if correct_option.save
        if imagepath.nil?
```

```
        Qa.create(questions_id: question.id, options_id: correct_option.id, exam_id: exam.id)
      else
        Qa.create(questions_id: question.id, options_id: correct_option.id, exam_id: exam.id, imagepath: imagepath)
      end

      redirect '/admin'
    end
  else
    @topics = Topic.all
    return erb :new_question, locals: { error: 'Failed to create question.' }
  end
end
```

Para solucionar esto, fue necesario realizar una modularización de varias funcionalidades específicas y agregarlas al archivo de helpers:

```
def render_new_question
  @topics = Topic.all
  erb :new_question, locals: { error: 'Failed to create question.' }
end

def save_image(params, topic_id)
  return nil unless Topic.find(topic_id).topic == 'Banderas' && params['image'][:filename].end_with?('.svg')

  filename = params['image'][:filename]
  file = params['image'][:tempfile]
  image_folder = './public/images/flags'
  filepath = "#{image_folder}/#{filename}"

  File.open(filepath, 'wb') do |f|
    f.write(file.read)
  end

  "images/flags/#{filename}"
end

def create_question_and_option(question_text, topic_id, correct_answer_text, exam_id, imagepath)
  question = Question.new(question: question_text, topic_id: topic_id)

  return unless question.save

  correct_option = Option.create(response: correct_answer_text, topics_id: topic_id)
  return unless correct_option.save

  Qa.create(questions_id: question.id, options_id: correct_option.id, exam_id: exam_id, imagepath: imagepath)
end
```

De esta manera, con métodos modularizados que realizan tareas específicas el endpoint que daba conflictos por su longitud pudo ser reducido en muchas líneas

```ruby
post '/admin/questions' do
    halt 403, 'Access denied.' unless current_user.admin?

    question_text, correct_answer_text, question_level, topic_id = params.values_at('question', 'correct_answer',
                                                                                    'question_level', 'topic_id')

    redirect '/admin/questions/new' if topic_id.nil?

    lesson = Lesson.find_by(topic_id: topic_id)
    level = Level.find_by(lesson_id: lesson.id, number: question_level)
    return render_new_question if level.nil?

    exam = Exam.find_by(lesson_id: lesson.id, level: level.id)

    imagepath = save_image(params, topic_id)

    if create_question_and_option(question_text, topic_id, correct_answer_text, exam.id, imagepath)
      redirect '/admin'
    else
      @topics = Topic.all
      return erb :new_question, locals: { error: 'Failed to create question.' }
    end
  end
```

como se aprecia el método se redujo en 33 líneas lo cual trae muchas ventajas y beneficios al proyecto como facilidad de comprensión, mantenimiento simplificado, reutilización de código, pruebas más efectivas, claridad en la intención, facilidad de refactorización, adherencia a principios de diseño, colaboración mejorada, menor riesgo de errores y optimización de la documentación.

3. Usar snake_case

   Antes del proceso de refactorización, habían nombres de variables que no seguían las convenciones de Ruby. Esto implicó renombrarlas para que adoptaran el estilo snake_case, en el que los espacios son reemplazados por guiones bajos, y todas las letras están en minúscula.

   En la siguiente situación de ejemplo, se cambió el nombre de la variable publicUser por public_user.

```ruby
get '/exam/:lesson_id/:level_id' do |lesson_id, level_id|
  @public_user = current_user.public_data
  @lesson = Lesson.find(lesson_id)
  @level = Level.find(level_id)
  @lesson_id = params[:lesson_id]
  @level_id = params[:level_id]
  redirect '/lessons/levels' unless level_unlocked?(@lesson, @level.number)
  @exam_id = Exam.find_by(lesson: @lesson, level: @level).id

  erb :quiz, locals: { lesson: @lesson, level: @level, exam: @exam }
end
```

```ruby
get '/exam/:lesson_id/:level_id' do |lesson_id, level_id|
  @publicUser = current_user.public_data
  @lesson = Lesson.find(lesson_id)
  @level = Level.find(level_id)
  @lesson_id = params[:lesson_id]
  @level_id = params[:level_id]
  if not level_unlocked?(@lesson, @level.number)
    redirect '/lessons/levels'
  end
  @exam_id = Exam.find_by(lesson: @lesson, level: @level).id

  erb :quiz, locals: { lesson: @lesson, level: @level, exam: @exam }
end
```

4. Líneas acotadas a un máximo de caracteres

   Por defecto, Rubocop limita la cantidad de caracteres en una línea a 120, por lo que tuvimos que dividir algunas líneas de código. En el siguiente ejemplo, se muestra como el código, que inicialmente estaba en una sola línea, se divide en dos para poder cumplir con este estilo.

```ruby
post '/signup', username: @user.username, email: @user.email, password: 'password', 'password-confirmation': 'password'
```

```ruby
    post '/signup', username: @user.username, email: @user.email, password: 'password',
                    'password-confirmation': 'password'
```

Conclusión

Comenzamos corriendo la gema Rubocop, y obteniendo un total de 1100 ofensas aproximadamente, algunas eran auto-corregidas por la misma gema y a otras, tuvimos que aplicar las reglas necesarias para obtener un código claro y de calidad, que reduce el esfuerzo de mantenimiento a largo plazo. Finalmente conseguimos un total de 0 ofensas.

Antes:

```
         ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

54 files inspected, 1101 offenses detected, 1041 offenses autocorrectable

Tip: Based on detected gems, the following RuboCop extension libraries might be helpful:
  * rubocop-rake (https://rubygems.org/gems/rubocop-rake)
  * rubocop-rspec (https://rubygems.org/gems/rubocop-rspec)
```

Después:

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

28 files inspected, no offenses detected

Tip: Based on detected gems, the following RuboCop extension libraries might be helpful:
  * rubocop-rake (https://rubygems.org/gems/rubocop-rake)
  * rubocop-rspec (https://rubygems.org/gems/rubocop-rspec)
```