Informe Refactorización
CulturAr
Sosa Fernando, Ignacio Dias Gundin

Code Smells

Identificamos en el código de nuestra aplicación muchos fallos en las buenas prácticas y en convenciones referidas a la programación con el lenguaje ruby, convenciones de espacios, limite del largo de una linea, los nombres de las variables y métodos tienen que respetar la escritura *snake_case*, todo estos fallos fueron corregidos facilmente por la herramienta de **RuboCop.**

Identificamos además que:

- todo el código estaba contenido en una sola clase, la clase “App” del server.rb

```ruby
# #Questions of Exams

get '/:category_name/levels/exam/:exam_id/questions/:question_id' do
  @catLvl = category_using_name(params[:category_name])
  question = Question.find_by(id: params[:question_id].to_i)
  exam = Exam.find(params[:exam_id])
  answers = [question.answer, question.wrongAnswer1, question.wrongAnswer2, question.wrongAnswer3].shuffle
  erb :question_exam, locals: { exam: exam, question: question, options: answers }
end

get '/:category_name/levels/exam/:exam_id/questions' do
  @catLvl = category_using_name(params[:category_name])
  questions = Question.where(exam_id:  params[:exam_id]).order('RANDOM()').pluck(:id)
  session[:questions_exam] = questions # Guardo el id de las preguntas
  if questions.empty?
    redirect to("/#{params[:category_name]}/levels")
  else
    first_question = questions.first

    redirect "/#{params[:category_name]}/levels/exam/#{params[:exam_id]}/questions/#{first_question}" # el URI.en
  end
end

get '/:category_name/levels/exam/:exam_id/completed' do
  @catLvl = category_using_name(params[:category_name])
  exam = RecordExam.find_by(exam_id: params[:exam_id])
  @totalPoints = exam.point
  erb :exam_completed # No hay más preguntas, mostrar mensaje de juego completado
end

get '/:category_name/levels/exam/:exam_id/fail' do
  @catLvl = category_using_name(params[:category_name])
  erb :exam_fail
end

post '/:category_name/levels/exam/:exam_id/questions/:question_id/resp' do
  @catLvl = category_using_name(params[:category_name])
  current_question = Question.find(params[:question_id])
  userAnswer = params[:userAnswer] # Obtener la respuesta enviada por el usuario
```

- había demasiados métodos en este mismo archivo server.rb en consecuencia de que todo lo contenía el mismo.

```ruby
# METODOS

def exam_finished(cat, exam_id)
  record = Record.find_by(user_id: session[:user_id])
  points_exam = complete_levels(cat)
  exam = Exam.find(exam_id)
  RecordExam.create(record_id: record.id, exam_id: exam.id, point: points_exam)
  update_points_profile(points_exam)
end

def complete_levels(cat)
  levels = Level.where(category_id: cat.id)
  points = 0
  levels.each do |lvl|
    next if RecordLevel.exists?(level_id: lvl.id)

    questions = Question.where(level_id: lvl.id)
    questions.each do |q|
      add_record_question(lvl.id, q, q.pointQuestion, true)
      points += q.pointQuestion
    end
    add_record_level(lvl)
  end
  points
end

def getPointLevel(level_id)
  record = Record.find_by(user_id: session[:user_id])
  record_level = RecordLevel.find_by(record_id: record.id, level_id: level_id)
  record_level.total_points
end

def levels_ids_completed
  user = User.find(session[:user_id])
  record = user.record
  RecordLevel.where(record_id: record.id).pluck(:level_id)
end

def update_points_profile(points)
```

- los **posts** de **questions** y **exams** estaban muy cargados de lógica y sin modularización.

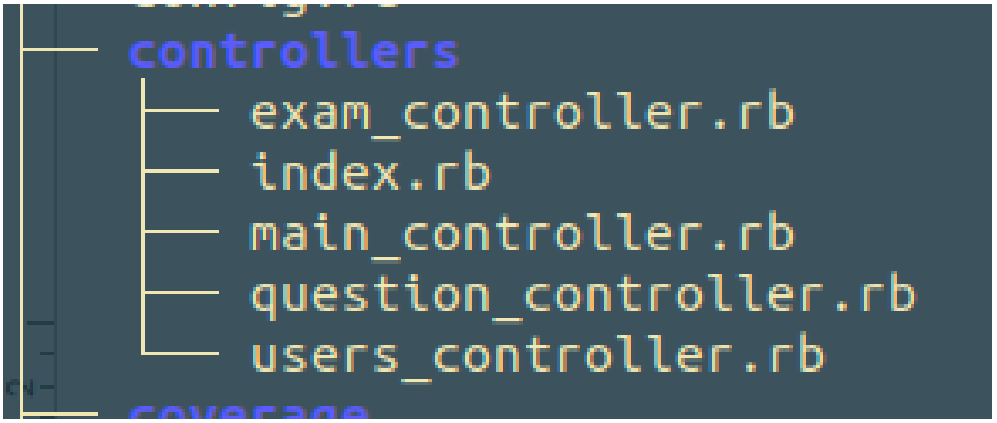
```ruby
post '/:category_name/levels/:level_id/questions/:question_id/resp' do
  @catLvl = category_using_name(params[:category_name])
  current_question = Question.find(params[:question_id])
  level = Level.find_by(id: params[:level_id])
  userAnswer = params[:userAnswer] # Obtener la respuesta enviada por el usuario
  if userAnswer.downcase == current_question.answer.downcase # Verificar si la respuesta es correcta
    current_point = current_question.pointQuestion # Cargo el registro de la pregunta completado
    add_record_question(params[:level_id], current_question, current_point, true)
    update_points_profile(current_point) # actualizo los puntos en el perfil
    quest_next = next_question(level.id, current_question.id) # Siguiente pregunta
    if quest_next.nil?
      add_record_level(level) # Agrego el registro del level completado
      redirect "/#{params[:category_name]}/levels/#{params[:level_id]}/completed"
    else
      redirect "/#{params[:category_name]}/levels/#{params[:level_id]}/questions/#{quest_next.id}" # Se reini
    end
  else
    add_record_question(params[:level_id], current_question, -5, false)
    update_points_profile(-5) # actualizo los puntos en el perfil      You, ahora • Uncommitted changes
    question = Question.find(params[:question_id])
    [question.answer, question.wrongAnswer1, question.wrongAnswer2, question.wrongAnswer3].shuffle
    redirect "/#{params[:category_name]}/levels/#{params[:level_id]}/questions/#{params[:question_id]}" # La
  end
end

post '/:category_name/levels/:level_id/completed' do
```

Refactorización

Soluciones:

- Lo primero fue adoptar el patrón de Modelo Vista Controlador (MVC) para descomprimir la clase “App” del server.rb, creando una carpeta “Controller”, y 4 archivos, “users_controller”, “main_controller”, “question_controller” y “exam_controller”



De esta forma pudimos extraer métodos y funcionalidades que sobrecargaron la clase “App” a estas nuevas clases que hicimos.
De esta forma está la clase App ahora:

```ruby
class App < Sinatra::Application
  def initialize(_app = nil)
    super()
  end

  # set :root,  File.dirname(__FILE__)
  set :root, File.dirname(__FILE__)
  # set :views, File.expand_path('../views', settings.root)
  set :views, File.expand_path('views', __dir__)
  # set :views, proc { File.join(root, '/views') }
  set :public_folder, File.join(root, 'static')

  configure :production, :development do
    enable :logging
    logger = Logger.new($stdout)
    logger.level = Logger::DEBUG if development?
    set :logger, logger
  end

  configure :development do
    register Sinatra::Reloader
    after_reload do
      logger.info 'Reloaded'
    end
  end

  enable :sessions

  set :session_expire_after, 7200

  before do
    restricted_paths = ['/lobby', '/profile', '/:category_name/levels', '/:category_name/:level_name/questions']

    redirect '/showLogin' if restricted_paths.include?(request.path_info) && !session[:user_id]
    redirect '/lobby' if (request.path_info == '/' || request.path_info == '/showLogin') && session[:user_id]
  end

  use UsersController
  use MainController
  use ExamController
  use QuestionController
end
```

- Realizamos un extract method sobre los posts de “questions” y de “exams”, para modularizar, que sea más legible, mantenible y reusable.

```ruby
post '/:category_name/levels/:level_id/questions/:question_id/resp' do
  # @cat_lvl = Category.category_using_name(params[:category_name])
  current_question = Question.find(params[:question_id])
  level = Level.find_by(id: params[:level_id])
  user_answer = params[:user_answer] # Obtener la respuesta enviada por el usuario
  if user_answer.downcase == current_question.answer.downcase # Verificar si la respuesta es correcta
    answer_correct(params[:category_name], current_question, level)
  else
    answer_incorrect(params[:category_name], current_question, level)
  end
end

# METODOS
def answer_correct(cat_name, current_question, level)
  current_point = current_question.pointQuestion # Cargo el registro de la pregunta completado
  RecordQuestion.add_record_question(session[:user_id], params[:level_id], current_question, current_point, true)
  Profile.update_points_profile(current_point, session[:user_id]) # actualizo los puntos en el perfil
  quest_next = Question.next_question(session[:user_id], level.id, current_question.id) # Siguiente pregunta
  if quest_next.nil?
    RecordLevel.add_record_level(level, session[:user_id]) # Agrego el registro del level completado
    redirect "/#{cat_name}/levels/#{level.id}/completed"
  else
    redirect "/#{cat_name}/levels/#{level.id}/questions/#{quest_next.id}" # Se reinicia los puntos penalizados que s
  end
end

def answer_incorrect(cat_name, current_question, level)
  RecordQuestion.add_record_question(session[:user_id], level.id, current_question, -5, false)
  Profile.update_points_profile(-5, session[:user_id]) # actualizo los puntos en el perfil
  question = Question.find(current_question.id)
  [question.answer, question.wrongAnswer1, question.wrongAnswer2, question.wrongAnswer3].shuffle
  redirect "/#{cat_name}/levels/#{level.id}/questions/#{current_question.id}" # La respuesta es incorrecta, volver a
end
end
```

- Muchos métodos que usaban la mayoría de get y post que estaban en la clase “App” de server.rb los movimos respectivamente a las clases de los modelos donde pertenecen.
  por ejemplo:

```ruby
class RecordLevel < ActiveRecord::Base
  belongs_to :record
  belongs_to :level

  def self.levels_ids_completed(user_id)
    user = User.find(user_id)
    record = user.record
    RecordLevel.where(record_id: record.id).pluck(:level_id)
  end

  def self.complete_levels(cat, user_id)
    levels = Level.where(category_id: cat.id)
    points = 0
    levels.each do |lvl|
      questions = Question.where(level_id: lvl.id)
      questions.each do |q|
        RecordQuestion.add_record_question(user_id, lvl.id, q, q.pointQuestion, true)
        points += q.pointQuestion
      end
      RecordLevel.add_record_level(lvl, user_id)
    end
    points
  end
```