Taller Ingeniería de Software

Actividad Nro: 4
Refactorización

Integrantes:
Buil Delfina
Lopez Agustin
Peruchin Ivan

Universidad Nacional de Río Cuarto

Comenzamos el proceso de refactorización con la separación de rutas en nuestro archivo `server.rb`. Hemos identificado un code smell en este archivo, donde se presentó un caso de Bloaters. A medida que nuestro programa evolucionó, `server.rb` fue creciendo más y más, llegando a un tamaño muy poco legible, insostenible y poco escalable. La solución que decidimos aplicar fue la división de este archivo en controladores. Creamos una carpeta llamada `controllers`, donde se alojan 4 controladores, cada uno compuesto por rutas del `server.rb` las cuales reúnen características determinadas para estar dentro de este mismo.
Antes de aplicar esta solución, nuestro archivo se veía de la siguiente manera:

```ruby
class App < Sinatra::Application
  enable :sessions
  # Configuracion de la clave secreta de sesión
  set :session_secret, 'la_pelota_no_se_mancha'

  def initialize(app = nil)
    super()
  end
  set :root, File.dirname(__FILE__)
  set :views, Proc.new { File.join(root, 'views') }
  set :public_folder, File.dirname(__FILE__) + '/views'

  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
      puts 'Reloaded...'
      logger.info 'Reloaded!!!'
    end
  end


  get '/game/:id_question' do
    session[:tree] = true #de arbol vuelve a game
    if session[:user_id].nil?
      redirect '/' # Redirigir al inicio de sesión si la sesión no está activa
    end
    @total_questions = Question.count # Numero total de preguntas en el juego
    @id_question = params[:id_question].to_i  # id de la pregunta a preguntar
    @level_selected = params[:level].to_i # este parámetro level viene de el post /levels

    user_id = session[:user_id]
    @user = User.find(user_id)
```

```ruby
post '/buyMoreTime' do
  user_id = session[:user_id]
  coins_to_decrement = 20

  user = User.find(user_id)

  if user.coin >= coins_to_decrement
    user.update(coin: user.coin - coins_to_decrement)
    content_type :json
    { success: true, updatedCoins: user.coin }.to_json
  else
    content_type :json
    { success: false }.to_json
  end
end


post '/incorrectOptions' do
  user_id = session[:user_id]
  coins_to_decrement = 10

  question_id = session[:question_id]

  user = User.find(user_id)

  if user.coin >= coins_to_decrement
    user.update(coin: user.coin - coins_to_decrement)
    incorrect_options = Option.where(question_id: question_id, isCorrect: false).pluck(:id)
    content_type :json
    { success: true, updatedCoins: user.coin, incorrect_options: incorrect_options }.to_json
  else
    content_type :json
    { success: false }.to_json
  end
end
```

Algunas capturas del archivo `server.rb`.

Totalmente desorganizado (con un total de 605 líneas de código), lleno de métodos y líneas de código que se repiten en diversos lados, muy poco legible, limitación en la tarea de su mantenimiento, poco mantenible y reutilizable, entre otras características.
Propusimos, entonces, crear cuatro controladores que entre ellos reúnen todas las funcionalidades y rutas de nuestra aplicación. Estos fueron los siguientes: `Authentication Controller`, `Game Controller`, `Menu Controller` y `Store Controller`.

Dentro de `Authentication Controller` reunimos todas aquellas funcionalidades que referían al registro de un nuevo usuario, su login, logout y la validación de su email. Con respecto al `Game Controller`, este fue compuesto por las funcionalidades que resolvían cuestiones propias de la jugabilidad de la aplicación. Dentro de estas se encuentran las rutas para manejar el juego con sus preguntas, las respuestas del usuario, los niveles y rutas para comprar ayudas (tiempo y eliminar dos opciones incorrectas).
En `Menu Controller` incluimos las rutas que se le presentan al usuario una vez que ingresa a su cuenta, estas son: ranking, su perfil con funcionalidades para poder cambiar sus datos,

su árbol y la sección de practicar. Por último, dentro del `Store Controller` abarcamos todas las funcionalidades y rutas referidas a la tienda de la aplicación.

Luego de haber ejecutado esta división y haber resuelto el code smell identificado en nuestro archivo principal, este resultó ser bastante más legible, entendible y mantenible, justo lo que queríamos lograr.

```ruby
class App < Sinatra::Application
  use AuthenticationController
  use GameController
  use MenuController
  use StoreController

  enable :sessions
  # Configuracion de la clave secreta de sesión
  set :session_secret, 'la_pelota_no_se_mancha'

  def initialize(app = nil)
    super()
  end
  set :root, File.dirname(__FILE__)
  set :views, Proc.new { File.join(root, 'views') }
  set :public_folder, File.dirname(__FILE__) + '/views'

  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
      puts 'Reloaded...'
      logger.info 'Reloaded!!!'
    end
  end

  get '/' do
    erb :start
  end
end
```

`server.rb` luego de su refactorización, conteniendo únicamente 58 líneas de código.

La próxima tarea a llevar a cabo fue refactorizar cada uno de estos controladores y sus respectivos modelos.

A continuación, se presenta un análisis y estudio de cada uno de estos controladores, con un reporte de los problemas identificados manualmente y sus determinados cambios, mejoras y acciones llevadas a cabo para solucionarlos. Cabe mencionar, además, el uso de la herramienta Rubocop, la cual reportó problemas y reglas de estilos que pudimos solucionar.

Game Controller:

Respecto a este controlador, comenzamos corriendo la herramienta de rubocop, sin aplicar ninguna regla de refactorización. Luego de corregir los errores de estilos del propio lenguaje, como espaciados, comentarios, variables escritas con camelCase, etc, nos quedaron las ofensas más importantes:

```
Offenses:

controllers/game_controller.rb:4:1: C: Metrics/ClassLength: Class has too many lines. [173/100]
class GameController < Sinatra::Application ...
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
controllers/game_controller.rb:6:3: C: Metrics/BlockLength: Block has too many lines. [42/25]
  get '/game/:id_question' do ...
  ^^^^^^^^^^^^^^^^^^^^^^^^^^^
controllers/game_controller.rb:59:3: C: Metrics/BlockLength: Block has too many lines. [31/25]
  post '/game/:question_id' do ...
  ^^^^^^^^^^^^^^^^^^^^^^^^^^^^

1 file inspected, 3 offenses detected
```

La primera ofensa indica que la clase sigue teniendo demasiadas líneas, principalmente los métodos **`get`** y **`post`** de la ruta **`/game`**. Para corregir eso se realizó lo siguiente:

- Se sacó del controlador consultas directas a la base de datos y se llevaron al modelo correspondiente. Utilizando la técnica de *Extract Method* para modularizar y también el *principio de responsabilidad única* para que el modelo sea el encargado de manipular la base de datos. Por ejemplo:

Antes:

```
# consultamos si esa pregunta habia sido preguntada
asked_question = AskedQuestion.find_by(user_id: user_id, question_id: @id_question)
```

Después:

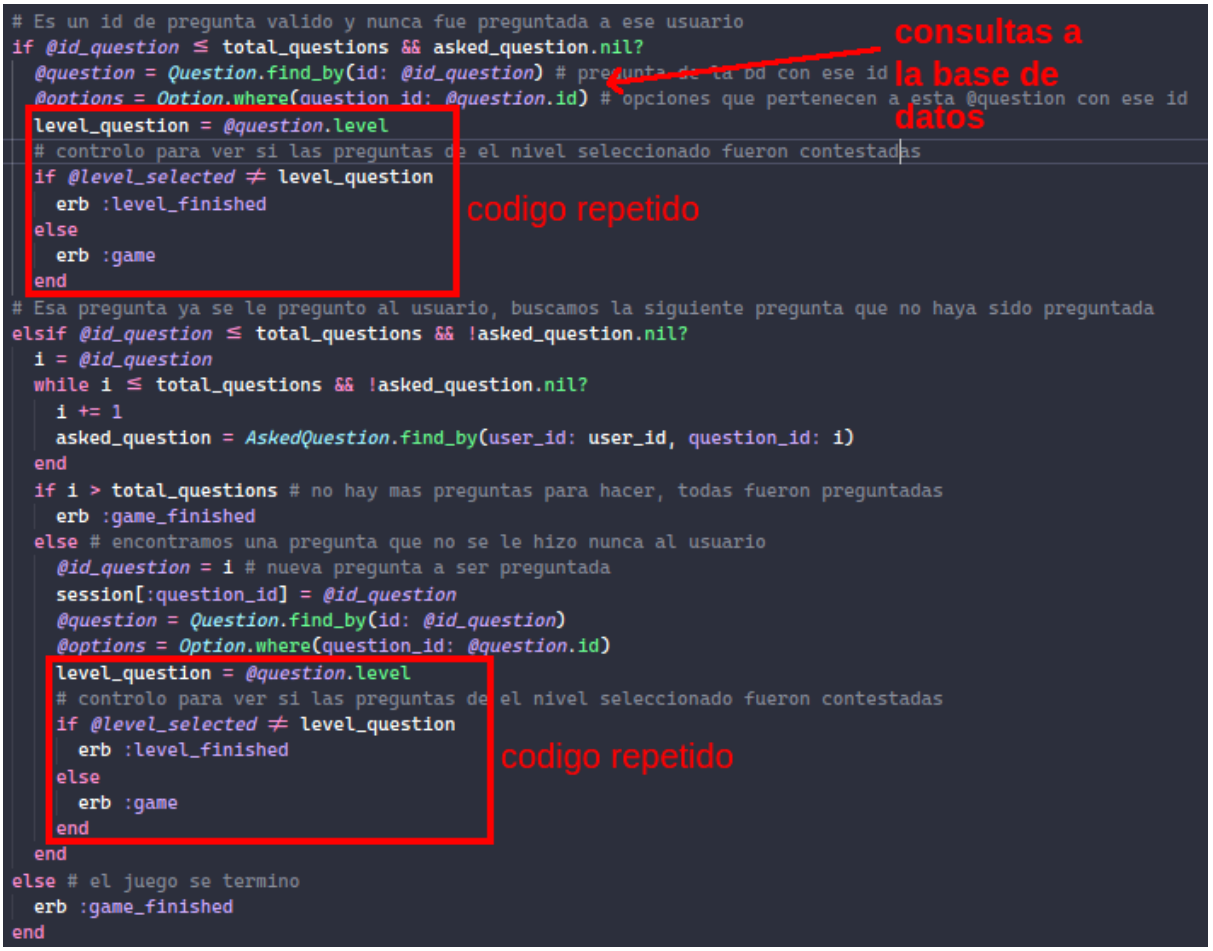
```
# consultamos si esa pregunta habia sido preguntada
asked_question = asked_question(@user.id, @id_question)
```

En el modelo:

```
def self.asked_question(user_id, question_id)
    record = AskedQuestion.find_by(user_id: user_id, question_id: question_id)
    return !record.nil? # True si se respondio, False si no
end
```

A este método también se le agregó pequeña lógica para simplificar código en el controlador.

- Luego, en el siguiente bloque de código, dentro del método **`get /game`** podemos enumerar varias correcciones:




```ruby
# Es un id de pregunta valido y nunca fue preguntada a ese usuario
if @id_question ≤ total_questions && asked_question.nil?
  @question = Question.find_by(id: @id_question) # pregunta de la bd con ese id
  @options = Option.where(question_id: @question.id) # opciones que pertenecen a esta @question con ese id
  level_question = @question.level
  # controlo para ver si las preguntas de el nivel seleccionado fueron contestadas
  if @level_selected ≠ level_question
    erb :level_finished
  else
    erb :game
  end
# Esa pregunta ya se le pregunto al usuario, buscamos la siguiente pregunta que no haya sido preguntada
elsif @id_question ≤ total_questions && !asked_question.nil?
  i = @id_question
  while i ≤ total_questions && !asked_question.nil?
    i += 1
    asked_question = AskedQuestion.find_by(user_id: user_id, question_id: i)
  end
  if i > total_questions # no hay mas preguntas para hacer, todas fueron preguntadas
    erb :game_finished
  else # encontramos una pregunta que no se le hizo nunca al usuario
    @id_question = i # nueva pregunta a ser preguntada
    session[:question_id] = @id_question
    @question = Question.find_by(id: @id_question)
    @options = Option.where(question_id: @question.id)
    level_question = @question.level
    # controlo para ver si las preguntas de el nivel seleccionado fueron contestadas
    if @level_selected ≠ level_question
      erb :level_finished
    else
      erb :game
    end
  end
else # el juego se termino
  erb :game_finished
end
```

Como se puede observar, se detectó código repetido y consultas a la base de datos cuya responsabilidad se podría estar delegando a sus correspondientes modelos.
Utilizando la técnica de *extraer método*, se puede modularizar para que el código resulte más simple y claro.

Resultado del método **`get /game`** luego de su refactorización:

```ruby
get '/game/:id_question' do
  session[:tree] = true
  question_id = params[:id_question].to_i
  @level_selected = params[:level].to_i
  total_questions = Question.total_questions
  redirect '/levels' if question_id > total_questions
  new_question = Question.find_next_question(question_id, @user.id, total_questions)
  if new_question.nil?
    erb :game_finished
  else
    @question, @options = new_question.values_at(:question, :options)
    session[:question_id] = @question.id
    erb(@level_selected ≠ @question.level ? :level_finished : :game)
  end
end
```

- Se extrajo el método “`find_next_question`” y se lo trasladó al modelo de “`Question`”:

```ruby
def self.find_next_question(current_question_id, user_id, total_questions)
  next_question_id = current_question_id
  while next_question_id ≤ total_questions
    if !AskedQuestion.asked_question(user_id, next_question_id)
      question = Question.find_question(next_question_id)
      options = Question.find_options(next_question_id)
      return { question: question, options: options }
    end
    next_question_id += 1
  end
  return nil
end
```

Lo cual permitió extraer gran parte de la lógica del controlador.

- Se realizó el método `before` para evitar repetir código en cada ruta.

```ruby
before do
  redirect '/' if session[:user_id].nil? && request.path_info ≠ '/'
  @user = User.current_user(session[:user_id]) unless session[:user_id].nil?
end
```

- Luego, en el método post de **‘/game’** se detectó lo siguiente:

```ruby
post '/game/:question_id' do
  user_id = session[:user_id]
  level = params[:level]
  if params[:selected_option_id].nil? && params[:timeout] == 'false'
    question_id = params[:question_id]
    redirect "/game/#{question_id}"
  end
  if params[:selected_option_id].nil? && params[:timeout] == 'true'
    AskedQuestion.create(user_id: user_id, question_id: params[:question_id])
    option_result = 'nil'
    selected_option_id = 999_999
    redirect "/asked/#{params[:question_id]}/#{option_result}/#{selected_option_id}?level=#{level}"
  end
  selected_option = Option.find(params[:selected_option_id])
  option_result = selected_option.isCorrect ? 'true' : 'false'
  # Calculo puntos del usuario
  user = User.find(user_id)
  if AskedQuestion.find_by(user_id: user_id, question_id: params[:question_id]).nil?
    if option_result == 'true'
      user.sum_points
      user.sum_streak
      user.sum_10_coins
      if (user.streak % 3).zero?
        user.add_streak_to_points(user.streak / 3)
        user.add_coins_from_streak((user.streak / 3) * 10)
      end
    else
      user.reset_streak
    end
    Answer.create(user_id: user_id, option_id: params[:selected_option_id])
    AskedQuestion.create(user_id: user_id, question_id: params[:question_id])
  end
  redirect "/asked/#{params[:question_id]}/#{option_result}/#{params[:selected_option_id]}?level=#{level}"
end
```


Dentro de este método, luego de un gran análisis, se pudo observar que este contenía bloques de código inalcanzables. En particular el primer bloque condicional ‘if’ (ubicado en las líneas 29-32), que contemplaba el caso en que el usuario haya presionado el botón responder (al estar jugando el juego) sin antes haber seleccionado una opción como respuesta. Este caso es controlado por medio de JavaScript en la sección de scripts de

nuestro proyecto, lo cual resulta innecesario realizar este control de nuevo en este controlador.
Asimismo, se pueden detectar más líneas de código innecesarias que lo único que aportan es una mayor complejidad al código.

Dentro del modelo `User`, se realizó un nuevo método utilizando la técnica "*Extract method*":

```ruby
def update_points
  self.sum_points
  self.sum_streak
  self.sum_10_coins
  if (self.streak % 3).zero?
    self.add_streak_to_points(self.streak / 3)
    self.add_coins_from_streak((self.streak / 3) * 10)
  end
end
```

Como así también dos nuevos métodos que se pueden observar a continuación:

```ruby
def self.createAskedQuestion(user_id, question_id)
    AskedQuestion.create(user_id: user_id, question_id: question_id)
end
```

```ruby
def self.createAnswer(user_id, option_id)
  Answer.create(user_id: user_id, option_id: option_id)
end
```

Con esto se delegó la responsabilidad de manipular la base de datos a los modelos correspondientes, en este caso el modelo `User`.

Como resultado final el método **`post '/game'`** resultó ser el siguiente:

```ruby
post '/game/:question_id' do
  level = params[:level]
  selected_option_id = params[:selected_option_id]
  question_id = params[:question_id].to_i
  if selected_option_id.nil? && params[:timeout] == 'true'
    AskedQuestion.createAskedQuestion(@user.id, question_id)
    redirect "/asked/#{question_id}/nil/nil?level=#{level}"
  end
  selected_option = Option.find(selected_option_id)
  # Calculo puntos del usuario si no la respondio nunca
  unless AskedQuestion.asked_question(@user.id, question_id)
    selected_option.isCorrect ? @user.update_points : @user.reset_streak
    Answer.createAnswer(@user.id, selected_option_id)
    AskedQuestion.createAskedQuestion(@user.id, question_id)
  end
  redirect "/asked/#{question_id}/#{selected_option.isCorrect}/#{selected_option_id}?level=#{level}"
end
```

- Se creó una carpeta separada para los scripts usados en el juego para extraer el código de JavaScript de la vista "`Game.erb`". (Técnica extract method)
- En la ruta **`get '/levels'`**, no se encontró mucho por refactorizar, solamente ciertas líneas innecesarias al haber agregado el método before y una consulta a la base de datos.

```ruby
get '/levels' do
  user = User.current_user(session[:user_id])
  @disabled_level = params[:disabled_level]
  @reset = params[:reset]
  @points = @user.points
  @levels = Question.distinct.pluck(:level)
  erb :levels
end
```

Código innecesario

Consulta directa a la base de datos

Quedando como resultado final:

```ruby
get '/levels' do
  @disabled_level = params[:disabled_level]
  @reset = params[:reset]
  @points = @user.points
  @levels = Question.get_all_levels
  erb :levels
end
```

- En el método **`post`** de **`'/levels'`**, antes de hacer modificaciones se encontraba de la siguiente manera:

```ruby
post '/levels' do
  user_id = session[:user_id]
  level = params[:levelSelected]
  question = Question.where(level: level).first()
  # obtengo la última pregunta del nivel anterior
  previous_question = question.id - 1
  if previous_question <= 0
    redirect "/game/#{question.id}?level=#{level}"
  end
  disabled_level = true
  response = AskedQuestion.find_by(user_id: user_id, question_id: previous_question).nil?

  # contro si puede acceder al nivel
  if response # response tiene true porque no encontró respuesta
    redirect "/levels?disabled_level=#{disabled_level}"
  else
    disabled_level = false
    redirect "/game/#{question.id}?level=#{level}"
  end
end
```

Donde se puede ver más de lo mismo, código que ya no es necesario por el método `'before'`.

Consultas a la base de datos que fueron trasladadas a sus respectivos modelos. En este caso, el método usado en **`'/game'`** llamado '`asked_question`' es reutilizado aquí.

También el uso de la variable '`disabled_level`' se consideró innecesario, basando nuestra opinión en la técnica "*Inline Temp*", que declara no usar variables para guardar un solo valor.

Como resultado se obtuvo lo siguiente:

```ruby
post '/levels' do
  level = params[:levelSelected]
  question = Question.first_question_level(level)
  previous_question = question.id - 1
  redirect "/game/#{question.id}?level=#{level}" if previous_question ≤ 0
  response = AskedQuestion.asked_question(@user.id, previous_question)
  response ? (redirect '/levels?disabled_level=true') : (redirect "/game/#{question.id}?level=#{level}")
end
```

Código más simple, muchas menos líneas.

En el método **`get`** de **`'/asked`**' antes de hacer modificaciones se veía así:

```ruby
get '/asked/:question_id/:option_result/:selected_option_id' do
  redirect '/' if session[:user_id].nil?
  @question = Question.find(params[:question_id])
  @user = User.find(session[:user_id])
  @result = params[:option_result]
  @level = params[:level]
  if @result == 'nil'
    @answer = 'Respuesta no contestada'
  else
    selected_option = Option.find(params[:selected_option_id])
    @answer = selected_option.description
  end

  @streak = @user.streak

  @correct = Option.find_by(isCorrect: 1, question_id: params[:question_id])&.description
  @respuesta = if @result == 'true'
                 'RESPUESTA CORRECTA'
               elsif @result == 'false'
                 'RESPUESTA INCORRECTA'
               else
                 'SE AGOTO EL TIEMPO'
               end
  erb :asked
end
```

En este caso no se pudo hacer demasiado por los distintos casos que se pueden dar, pero si se realizó un mejor manejo de condiciones para no repetir dos veces el condicional con la variable “`@result`”.

Así se ve el método resultante:

```ruby
get '/asked/:question_id/:option_result/:selected_option_id' do
  @question = Question.find_question(params[:question_id])
  @result = params[:option_result]
  @level = params[:level]
  if @result == 'true'
    @respuesta = 'RESPUESTA CORRECTA'
    @user_answer = Option.find(params[:selected_option_id]).description
  elsif @result == 'false'
    @respuesta = 'RESPUESTA INCORRECTA'
    @user_answer = Option.find(params[:selected_option_id]).description
  else
    @user_answer = 'Pregunta no contestada'
    @respuesta = 'SE AGOTO EL TIEMPO'
  end
  @streak = @user.streak
  @correct_answer = Option.find_correct_option(@question.id)
  erb :asked
end
```

- Dentro de la ruta **`/buyMoreTime`** se aplicaron las siguientes técnicas:

```
post '/buyMoreTime' do
  user_id = session[:user_id]
  coins_to_decrement = 30

  user = User.find(user_id)

  if user.coin >= coins_to_decrement
    user.update(coin: user.coin - coins_to_decrement)
    content_type :json
    { success: true, updatedCoins: user.coin }.to_json
  else
    content_type :json
    { success: false }.to_json
  end
end
```

Ruta antes de la refactorización.

Se reemplazó la actualización de las monedas del usuario por el uso del método `discount_coins` dentro del modelo User. Anteriormente se hacía una actualización directa a la base de datos, ahora el modelo User es quien se encarga de realizar esta actualización permitiendo una mejor abstracción y modularización. Esto mismo se aplicó en la ruta **`/incorrectOptions.`**
Además, se reemplazó la búsqueda directa en la base de datos del usuario según su id al utilizar el método `current_user` del modelo `User`.

```
post '/buyMoreTime' do
  coins_to_decrement = 30
  if @user.coin >= coins_to_decrement
    @user.discount_coins(coins_to_decrement)
    content_type :json
    { success: true, updatedCoins: @user.coin }.to_json
  else
    content_type :json
    { success: false }.to_json
  end
end
```

Resultado final.

- Dentro de la ruta **`/incorrectOptions`** se extrajo una operación que se estaba llevando a cabo dentro del controlador, cuando en realidad debería estar modularizada dentro del modelo Options. Esta operación obtiene las opciones incorrectas de una pregunta

en particular. Esto fue modularizado en un método llamado `incorrect_options` dentro del modelo Options para una mejor abstracción y reusabilidad.

```
post '/incorrectOptions' do
  user_id = session[:user_id]
  coins_to_decrement = 10

  question_id = session[:question_id]

  user = User.find(user_id)

  if user.coin >= coins_to_decrement
    user.update(coin: user.coin - coins_to_decrement)
    incorrect_options = Option.where(question_id: question_id, isCorrect: false).pluck(:id)
    content_type :json
    { success: true, updatedCoins: user.coin, incorrect_options: incorrect_options }.to_json
  else
    content_type :json
    { success: false }.to_json
  end
end
```

Antes.

```
post '/incorrectOptions' do
  coins_to_decrement = 10
  question_id = session[:question_id]

  if @user.coin >= coins_to_decrement
    @user.discount_coins(coins_to_decrement)
    incorrect_options = Option.incorrect_options(question_id)
    content_type :json
    { success: true, updatedCoins: @user.coin, incorrect_options: incorrect_options }.to_json
  else
    content_type :json
    { success: false }.to_json
  end
end
```

Después.

```
def self.incorrect_options(id)
  return Option.where(question_id: id, isCorrect: false).pluck(:id)
end
```

Método dentro del modelo Option.

- En la ruta **`get '/play'`** se puede observar código innecesario causado por el agregado del método `before` y consultas a la base de datos con métodos ya usados antes. En parte del código se puede utilizar la técnica de "*Extract Method*" para realizar las funcionalidades de reseteo de las respuestas y puntos del usuario.

Así se presentaba el método antes de la refactorización:

```ruby
get '/play' do
  redirect '/' if session[:user_id].nil?

  user_id = session[:user_id]
  @user = User.find(user_id)

  @total_questions = Question.count
  @user.update(points: 0)

  @user.reset_streak

  i = 1
  while i ≤ @total_questions
    asked_question = AskedQuestion.find_by(user_id: user_id, question_id: i)
    asked_question&.destroy
    i += 1
  end
  reset = true
  redirect "/levels?reset=#{reset}"
end
```

En este bloque puede verse claramente que el mismo puede ser trasladado dentro del modelo `User` con el objetivo de resetear su progreso, en vez de estar siendo resuelto dentro del controller:

```ruby
@user.update(points: 0)
@user.reset_streak
i = 1
while i ≤ @total_questions
  asked_question = AskedQuestion.find_by(user_id: user_id, question_id: i)
  asked_question&.destroy
  i += 1
end
```

Esto mismo se llevó a cabo a continuación:

```ruby
def reset_progress
  update(points: 0)
  reset_streak
  (1..Question.total_questions).each do |i|
    asked_question = AskedQuestion.find_by(user_id: self.id, question_id: i)
    asked_question&.destroy
  end
end
```

Método en el modelo `User`. Cumpliendo con las ofensas Rubocop.

Finalmente quedando el método **`get '/play'`** como se muestra a continuación:

```ruby
get '/play' do
  @user.reset_progress
  redirect '/levels?reset=true'
end
```

Menu Controller:

Dentro de este controlador, varios problemas se pudieron identificar y solucionar. Al ejecutar la herramienta Rubocop, esta reportó diferentes reglas de estilo que pudieron ser resueltas. El informe inicial que esta herramienta proporcionó contenía lo siguiente:

```
1 file inspected, 76 offenses detected, 50 offenses auto-correctable
```

Donde muchas de estas ofensas eran reglas de estilo propias del lenguaje de programación utilizado. Luego de una refactorización, se obtuvo un nuevo informe reportado por la herramienta, el cual resultó ser el siguiente:

```
1 file inspected, no offenses detected
```

Para lograr esto, se llevó a cabo lo siguiente:

- Dentro de la ruta **`/practicar`**:

```ruby
get '/practicar' do


  user_id = session[:user_id]
  @user = User.find(user_id)
  # array de todas los id de las preguntas que se le hicieron al usuario
  @questions_asked = AskedQuestion.where(user_id: user_id).pluck(:question_id)
  session[:questions_asked] = @questions_asked

  #indice de la current question a practicar
  @question_index = 0
  session[:question_index] = @question_index
  erb :practice
end
```

Se aplicó la técnica: *Extract Method*, ya que se observó la posibilidad de modularizar esta ruta para lograr que sea más legible y reutilizable.

```ruby
get '/practicar' do
  user_id = session[:user_id]
  setup_practice_data(user_id)
  erb :practice
end

def setup_practice_data(user_id)
  @questions_asked = AskedQuestion.where(user_id: user_id).pluck(:question_id)
  session[:questions_asked] = @questions_asked

  @question_index = 0
  session[:question_index] = @question_index
end
```

- Se modularizó la funcionalidad de encontrar el usuario que inició la sesión que se encuentra activa. Este nuevo método fue creado en el modelo User, y se utiliza ahora en las rutas: **`/menu`**, **`/profile_change`**, **`/ranking`**, **`/profile`** y **`/practicar`**.

```
def self.current_user(field, value)
  return User.find_by(field => value) if value && field
end
```

Funcionalidad modularizada en el modelo User.

```
get '/practicar' do
  user_id = session[:user_id]
  setup_practice_data(user_id)
  erb :practice
end
```

Ejemplo de nuevo método siendo utilizado en la ruta **`/practicar`**.

- Rubocop informó una regla de estilo que debía ser aplicada: los nombres de las variables deben estar escritos en formato snake_case. Este cambio se aplicó en la ruta **`/profile_change`**.

```
post '/profile_change' do
  user_id = session[:user_id]
  user = User.find_by(id: user_id)


  newUsername = params[:newUsername]
  currentPassword = params[:currentPassword]
  newPassword = params[:newPassword]
  newEmail = params[:newEmail]
```

```
post '/profile_change' do
  user_id = session[:user_id]

  new_username = params[:new_username]
  current_password = params[:current_password]
  new_password = params[:new_password]
  new_email = params[:new_email]
```

- Otro reporte hecho por la herramienta fue la estructura de los condicionales en la ruta **`/profile_change`.**

```ruby
if newUsername != "" && (!User.find_by(username: newUsername).nil?)
  redirect '/profile_change'
end
if newPassword != "" && !currentPassword.nil? && (currentPassword != user.password )
  redirect '/profile_change'
end
if newEmail != "" && (!User.find_by(email: newEmail).nil?)
  redirect '/profile_change'
end

if newUsername != ""
  user.update_column(:username, newUsername)
end

if newPassword != "" && currentPassword != ""
  user.update_column(:password, newPassword)
end


if newEmail != ""
  user.update_column(:email, newEmail)
end
```

Resultado:

```ruby
if (new_username != '' && !User.find_by(username: new_username).nil?) ||
   (new_password != '' && !current_password.nil? && (current_password != @user.password)) ||
   (new_email != '' && !User.find_by(email: new_email).nil?)
  redirect '/profile_change'
end
```

```ruby
@user.update_column(:username, new_username) if new_username != ''

@user.update_column(:password, new_password) if new_password != '' && current_password != ''

@user.update_column(:email, new_email) if new_email != ''
```

De esta manera, se consolidan tres secciones en una sola estructura condicional `if`, reduciendo así la repetición de código ya que su bloque de ejecución era el mismo. Además, se hace más legible el resto de los condicionales donde el bloque de ejecución era muy simple (como en las últimas tres estructuras condicionales).

- Luego de haber modularizado la funcionalidad para hallar el usuario que se encuentra con la sesión activa, se pudo detectar que había una variable en ciertas rutas, que no estaba siendo utilizada en ningún sitio.
  Esto sucedió en las rutas **`/tree`**, **`/practicar`** y **`/profile_change`**. La herramienta Rubocop determinó lo siguiente en base a este problema: *“Useless assignment to variable”*.

```ruby
get '/tree' do
  if session[:user_id].nil?
    redirect '/' # Redirigir al inicio de sesión si la sesión no está activa
  end
  user_id = session[:user_id]
  @user = User.find(user_id)
  hoja_id = @user.leaf_id #Busco el id de la actual compra del usuario
  fondo_id = @user.background_id
  @tree = session[:tree]
  @hoja = Item.find_by(id: hoja_id).name  #con ese id busco en los items y paso el nombre
  @fondo = Item.find_by(id: fondo_id).name
  erb :tree
end
```

```ruby
get '/tree' do
  user_id = session[:user_id]

  hoja_id = @user.leaf_id
  fondo_id = @user.background_id
  @hoja = Item.find_by(id: hoja_id).name
  @fondo = Item.find_by(id: fondo_id).name

  @tree = session[:tree]
  erb :tree
end
```

- Se logró también modularizar dos operaciones que se repetían en todos los métodos, utilizando la técnica *Extract Method.* Estas operaciones eran las siguientes:

```ruby
get '/menu' do
  session[:tree] = false
  if session[:user_id].nil?
    redirect '/' # Redirigir al inicio de sesión si la sesión no está activa
  end
  user_id = session[:user_id]
  @user = User.find(user_id)
  erb :menu, :locals => {:user_id => user_id}
end
```

Resultado final:

```ruby
before do
  redirect '/' if session[:user_id].nil? && request.path_info != '/'
  @user = User.current_user(session[:user_id]) unless session[:user_id].nil?
end
```

Este método se ejecuta antes de cualquier otra ruta permitiendo que un usuario no pueda acceder a las distintas secciones del juego mediante un URL, sin previamente haber iniciado sesión en el juego. Además, ahora todas las rutas harán referencia a la variable `@user` como el usuario actual que está iniciado en el sistema.

Store Controller:

Dentro de este controller se pudieron hallar diversos problemas, entre estos se encuentra la duplicación de código. Las vistas referidas a las funcionalidades de compra de hojas y compra de fondos brindadas al usuario, presentaban el mismo código, con algunas leves modificaciones.
Para resolver esta situación, se hizo uso de la técnica *"Mover Funcionalidades entre Objetos"*, donde se logró situar estas funcionalidades repetidas en una misma clase.

```ruby
get '/buyItem' do
  user_id = session[:user_id]
  @user = User.find(user_id)
  @coin = @user.coin
  @item_selected = params[:item]
  @item = Item.where(section: @item_selected)

  if (@item_selected == 'hoja')
    @item_selected_id = @user.leaf_id
  elsif (@item_selected == 'fondo')
    @item_selected_id = @user.background_id
  end

  purchased_item_ids = PurchasedItem.where(user_id: user_id).pluck(:item_id)

  @item_comprados = {}
  @item_price = {}
  @item.each do |item|
    @item_comprados[item.id] = purchased_item_ids.include?(item.id)
    @item_price[item.id] = item.price
  end

  erb :buyItem
end

post '/buyItem' do
  user_id = session[:user_id]
  user = User.find_by(id: user_id)
  item_selected = params[:item]

  request_body = JSON.parse(request.body.read)
  name = request_body['name']
  item_id = Item.find_by(name: name).id
  item_price = Item.find_by(name: name).price

  user.buy_item(item_id)
  user.set_item_in_user(item_selected, item_id)
```

Captura de `Store_controller` antes de realizarle cambios.

Quedando así un código unificado para una misma funcionalidad, mucho más prolijo y legible.
Se identificó, además, ciertos métodos que manipulan la base de datos dentro de este controler, que en efecto deberían estar siendo resueltos dentro del modelo `User`. A continuación se presenta el modelo `User` con sus nuevas funcionalidades agregadas:

```
def set_item_in_user(item, item_id)
  if (item == 'hoja')
    self.update_column(:leaf_id, item_id)
  else
    self.update_column(:background_id, item_id)
  end
  save
end

def buy_item(item_id)
  if PurchasedItem.find_by(item_id: item_id, user_id: self.id).nil?
    PurchasedItem.create(user_id: self.id, item_id: item_id)

    if (item_price <= self.coin)
      self.discount_coins(item_price)
    end

  end
  save
end
```

Esto se llevó a cabo aplicando la técnica "*Move Method*".

Otra situación analizada fue la repetida consulta para obtener el usuario que se encuentra momentáneamente inicializado en su sesión. Esto se puede observar a continuación:

```
get '/store' do

  user_id = session[:user_id]
  @user = User.find(user_id)
  @coin = @user.coin
  erb :store
end
```

Esto mismo ocurre en la ruta **`/buyItem`** como se puede visualizar en la siguiente imagen:

```ruby
get '/buyItem' do
  user_id = session[:user_id]
  @user = User.find(user_id)
  @coin = @user.coin
  @item_selected = params[:item]
  @item = Item.where(section: @item_selected)

  if (@item_selected == 'hoja')
    @item_selected_id = @user.leaf_id
  elsif (@item_selected == 'fondo')
    @item_selected_id = @user.background_id
  end

  purchased_item_ids = PurchasedItem.where(user_id: user_id).pluck(:item_id)

  @item_comprados = {}
  @item_price = {}
  @item.each do |item|
    @item_comprados[item.id] = purchased_item_ids.include?(item.id)
    @item_price[item.id] = item.price
  end

  erb :buyItem
end

post '/buyItem' do
  user_id = session[:user_id]
  user = User.find_by(id: user_id)
  item_selected = params[:item]

  request_body = JSON.parse(request.body.read)
  name = request_body['name']
  item_id = Item.find_by(name: name).id
  item_price = Item.find_by(name: name).price

  user.buy_item(item_id)
  user.set_item_in_user(item_selected, item_id)
```

En violeta se muestra el código repetido

Se está realizando una consulta directa a la base de datos, lo cual no debería estar sucediendo dentro de este controller.
Haciendo uso de la técnica "*Extract Method*", se logró modularizar esta funcionalidad en un método que lo resuelva, definido dentro del modelo `User`.

```ruby
# Obtiene el usuario actual en la sesion.
def self.current_user(field, value)
  return User.find_by(field => value) if value and field
end
```

Método `current_user` definido en modelo `User`.

Este fue utilizado en las rutas **"`/store`"** y **"`/buyItem`"**. Se observaban, también, variables donde se guardaba un solo valor, como se muestra a continuación:

```ruby
get '/buyItem' do
  user_id = session[:user_id]
  @user = User.find(user_id)
  @coin = @user.coin
  @item_selected = params[:item]
  @item = Item.where(section: @item_selected)

  if (@item_selected == 'hoja')
    @item_selected_id = @user.leaf_id
  elsif (@item_selected == 'fondo')
    @item_selected_id = @user.background_id
  end

  purchased_item_ids = PurchasedItem.where(user_id: user_id).pluck(:item_id)

  @item_comprados = {}
  @item_price = {}
  @item.each do |item|
    @item_comprados[item.id] = purchased_item_ids.include?(item.id)
    @item_price[item.id] = item.price
  end

  erb :buyItem
end

post '/buyItem' do
  user_id = session[:user_id]
  user = User.find_by(id: user_id)
  item_selected = params[:item]

  request_body = JSON.parse(request.body.read)
  name = request_body['name']
  item_id = Item.find_by(name: name).id
  item_price = Item.find_by(name: name).price

  user.buy_item(item_id)
  user.set_item_in_user(item_selected, item_id)
```

Recuadrado en verde se distinguen las variables temporales.

Situación que fue resuelta utilizando la técnica “*Inline Temp*”.
El resultado de la aplicación de estas técnicas se puede observar a continuación:

```ruby
get '/store' do
  @user = User.current_user(:id, (session[:user_id]))
  @coin = @user.coin
  erb :store
end

get '/buyItem' do
  @user = User.current_user(:id, (session[:user_id]))
  @coin = @user.coin
  @item_selected = params[:item]
  @item = Item.where(section: @item_selected)

  if @item_selected == 'hoja'
    @item_selected_id = @user.leaf_id
  elsif @item_selected == 'fondo'
    @item_selected_id = @user.background_id
  end

  purchased_item_ids = PurchasedItem.where(user_id: @user.id).pluck(:item_id)

  @item_comprados = {}
  @item_price = {}
  @item.each do |item|
    @item_comprados[item.id] = purchased_item_ids.include?(item.id)
    @item_price[item.id] = item.price
  end

  erb :buyItem
end
```

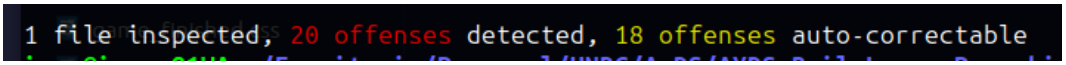
```ruby
post '/buyItem' do
  user = User.current_user(:id, (session[:user_id]))

  request_body = JSON.parse(request.body.read)
  name = request_body['name']
  item = Item.find_by(name: name)

  user.buy_item(item.id, item.price)
  user.set_item_in_user(params[:item], item.id)
end
```

Luego, al analizar nuevamente el código, se descubrió que en cada ruta, tanto en "**`/store`**" como en "**`/buyItem`**" se realiza la consulta del usuario, (se puede observar en las capturas anteriores), repitiendo así el mismo código. Para solucionarlo, se usó la técnica "*Extract Method*", donde se creó un nuevo método que encapsula esta funcionalidad y se ejecuta antes de todas las rutas, permitiendo que la variable `@user` esté disponible en todos los demás métodos:

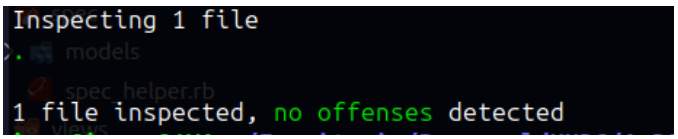
```ruby
# Esta es la clase del controlador de la tienda.
# Maneja las rutas de la tienda y la compra de hojas/fondos denominados Items.
class StoreController < Sinatra::Application
  before do
    redirect '/' if session[:user_id].nil? && request.path_info != '/'
    @user = User.current_user(:id, (session[:user_id])) unless session[:user_id].nil?``
  end
```

Con esto se logró eliminar la consulta del usuario que se repite en los métodos previamente mencionados.

Al ejecutar la herramienta Rubocop sobre este controller, se presentaban un total de 20 ofensas:



Luego de corregir ofensas automáticamente y de corregir las dos restantes manualmente (Documentación), se ejecutó nuevamente la herramienta y se obtuvo el siguiente informe:

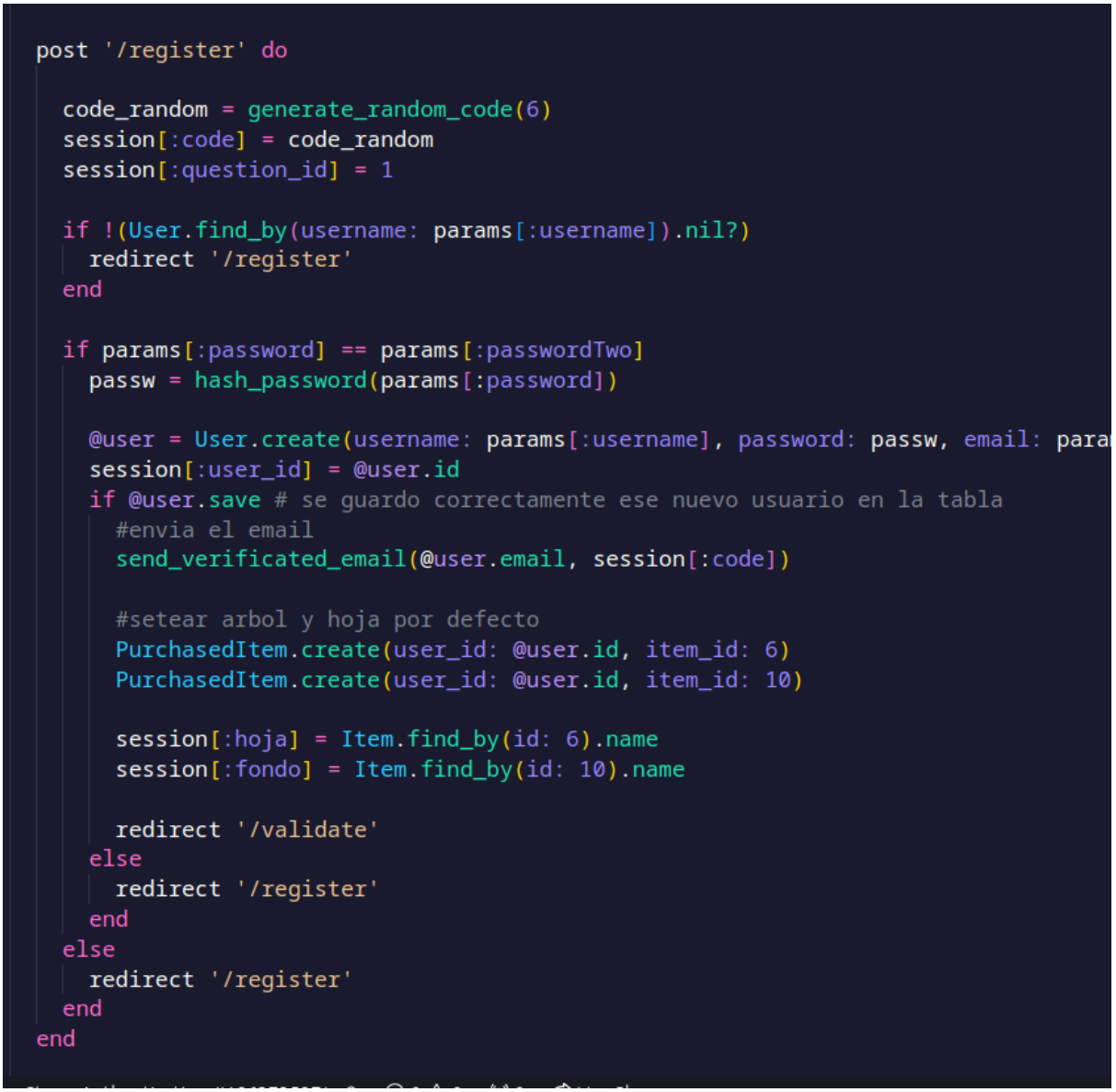


Authentication Controller:

Authentication Controller es un controlador en donde se encuentran las rutas para que un usuario pueda iniciar sesión y registrarse, validar su email y cerrar su sesión.
En este controller, se pudieron identificar problemas como la repetición de código y variables temporales.
Se verá más detalladamente a continuación cada problema:

Para facilitar la comparación, se presentan capturas de pantalla de la situación en la que se encontraban los métodos dentro de este controller:

```ruby
  post '/login' do
    @user = User.find_by(username: params[:username])
    input_password = params[:password]
    if @user && @user.compare_password(@user.password, input_password)
      session[:user_id] = @user.id

      session[:hoja] = Item.find_by(id: 6).name
      session[:fondo] = Item.find_by(id: 10).name
      redirect '/menu'
    elsif @user
        @password_error = "*contraseña incorrecta"
        erb :login
    else
      @password_error = "*datos ingresados no pertenecen a ninguna cuenta"
      erb :login
    end
  end

  get '/logout' do
    session.clear
    redirect '/'
  end

  get '/validate' do
    if session[:user_id].nil?
      redirect '/' # Redirigir al inicio de sesión si la sesión no está activa
    end
    erb :validate
  end

  post '/validate' do
    if session[:code] == params[:codigo]
      user = User.find(session[:user_id])
      user.update_column(:valid_email, true)
    end
    redirect '/menu'
  end
end
```



```ruby
post '/register' do

  code_random = generate_random_code(6)
  session[:code] = code_random
  session[:question_id] = 1

  if !(User.find_by(username: params[:username]).nil?)
    redirect '/register'
  end

  if params[:password] == params[:passwordTwo]
    passw = hash_password(params[:password])

    @user = User.create(username: params[:username], password: passw, email: para
    session[:user_id] = @user.id
    if @user.save # se guardo correctamente ese nuevo usuario en la tabla
      #envia el email
      send_verificated_email(@user.email, session[:code])

      #setear arbol y hoja por defecto
      PurchasedItem.create(user_id: @user.id, item_id: 6)
      PurchasedItem.create(user_id: @user.id, item_id: 10)

      session[:hoja] = Item.find_by(id: 6).name
      session[:fondo] = Item.find_by(id: 10).name

      redirect '/validate'
    else
      redirect '/register'
    end
  else
    redirect '/register'
  end
end
```

Capturas de pantalla donde se observa la situación en la que se encontraba este Controller.

Como se mencionó anteriormente, en este caso también se realizaban consultas directas a la base de datos para obtener el usuario actual con el que se está trabajando. Se pudo solucionar de la misma manera que en el controlador anterior, usando el método creado en el modelo `User`.

Otra situación observada es la repetida utilización de la funcionalidad para setear valores por defecto en la tienda del usuario.

```
      send_verificated_email(@user.email, session[:code])

      #setear arbol y hoja por defecto
      PurchasedItem.create(user_id: @user.id, item_id: 6)
      PurchasedItem.create(user_id: @user.id, item_id: 10)

      session[:hoja] = Item.find_by(id: 6).name
      session[:fondo] = Item.find_by(id: 10).name

      redirect '/validate'
    else
```

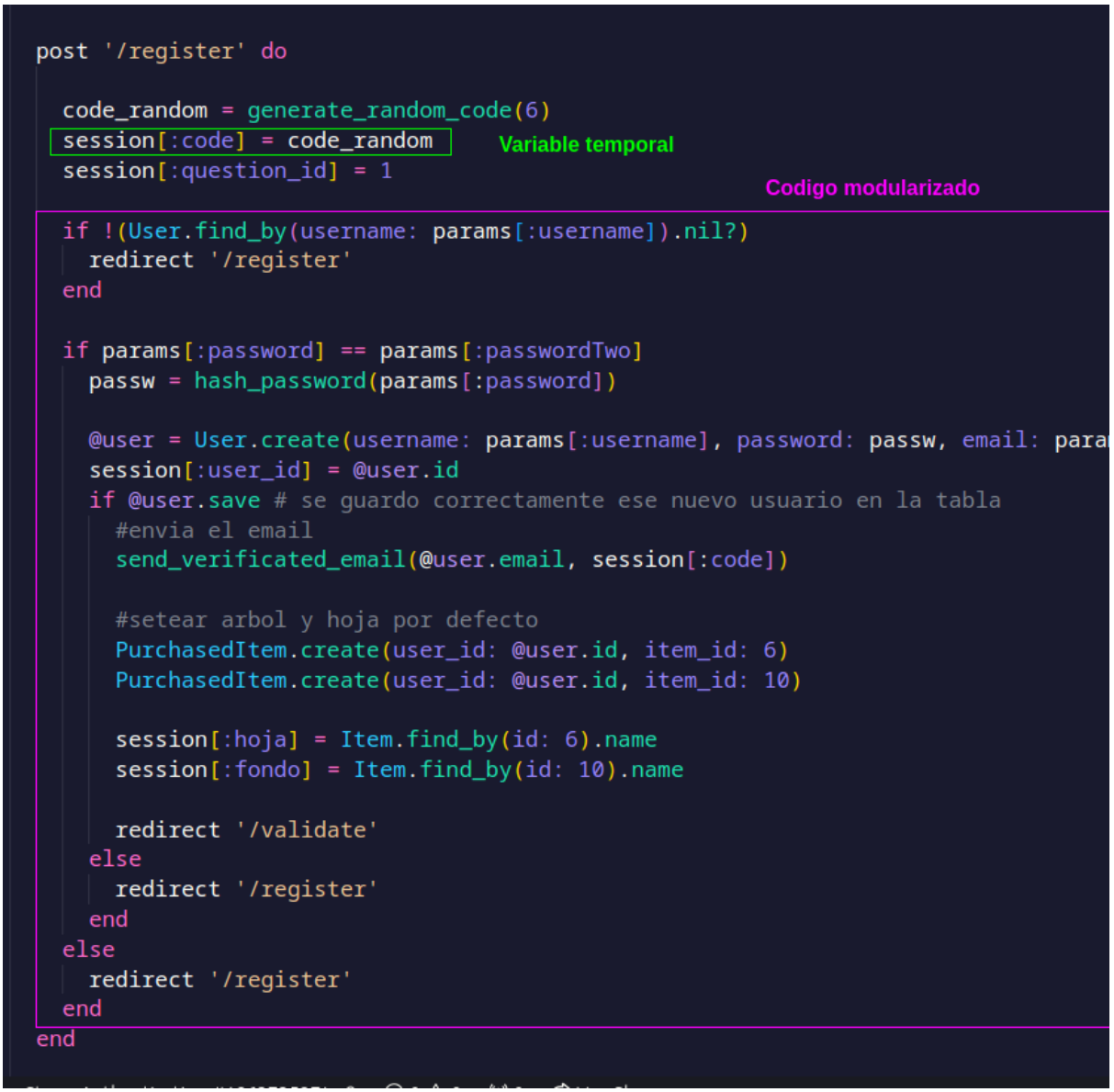
Se encuentra recuadrado en violeta el código que se repite en **`/register`** y **`/login.`**

Esto se resolvió mediante la técnica "*Extract Method*" donde se logró modularizar esta operación.

```
class Item < ActiveRecord::Base
  # Metodo para establecer una hoja y un fondo por defecto.
  def self.set_item_default(session)
    session[:hoja] = Item.find_by(id: 6).name
    session[:fondo] = Item.find_by(id: 10).name
  end
```

Este método se encuentra en el modelo `Item`.

Además, se modularizó la sección de código en donde se verifica que el usuario exista, en el caso de método **`/login,`** como también la funcionalidad que crea un nuevo usuario en el método **`/register`**.




```ruby
post '/register' do

  code_random = generate_random_code(6)
  session[:code] = code_random
  session[:question_id] = 1

  if !(User.find_by(username: params[:username]).nil?)
    redirect '/register'
  end

  if params[:password] == params[:passwordTwo]
    passw = hash_password(params[:password])

    @user = User.create(username: params[:username], password: passw, email: para
    session[:user_id] = @user.id
    if @user.save # se guardo correctamente ese nuevo usuario en la tabla
      #envia el email
      send_verificated_email(@user.email, session[:code])

      #setear arbol y hoja por defecto
      PurchasedItem.create(user_id: @user.id, item_id: 6)
      PurchasedItem.create(user_id: @user.id, item_id: 10)

      session[:hoja] = Item.find_by(id: 6).name
      session[:fondo] = Item.find_by(id: 10).name

      redirect '/validate'
    else
      redirect '/register'
    end
  else
    redirect '/register'
  end
end
```

Fragmento de código dentro del método **`/register`** que se modularizará.


```ruby
post '/login' do
  @user = User.find_by(username: params[:username])
  input_password = params[:password]
  if @user && @user.compare_password(@user.password, input_password)
    session[:user_id] = @user.id

    session[:hoja] = Item.find_by(id: 6).name
    session[:fondo] = Item.find_by(id: 10).name
    redirect '/menu'
  elsif @user
      @password_error = "*contraseña incorrecta"
      erb :login
  else
    @password_error = "*datos ingresados no pertenecen a ninguna cuenta"
    erb :login
  end
end

get '/logout' do
  session.clear
  redirect '/'
end

get '/validate' do
  if session[:user_id].nil?
    redirect '/' # Redirigir al inicio de sesión si la sesión no está activa
  end
  erb :validate
end

post '/validate' do
  if session[:code] == params[:codigo]
    user = User.find(session[:user_id])
    user.update_column(:valid_email, true)
  end
  redirect '/menu'
end
end
```

Fragmento de código dentro del método **`/login`** que se modularizará.

A continuación se presentan los dos nuevos métodos dentro del controller que abarcan estas dos funcionalidades.

```ruby
# Metodo para iniciar sesion.
def check_user(user)
  if user&.compare_password(user.password, params[:password])
    session[:user_id] = user.id
    Item.set_item_default(session)
    redirect '/menu'
  elsif user
    @password_error = '*contraseña incorrecta'
    erb :login
  else
    @password_error = '*datos ingresados no pertenecen a ninguna cuenta'
    erb :login
  end
end

# Metodo para registrar un usuario.
def create_user
  if passwords_match?
    user_attributes = {
      username: params[:username],
      password: hash_password(params[:password]),
      email: params[:email],
      birthdate: params[:birthdate],
      leaf_id: 6,
      background_id: 10
    }

    @user = User.create(user_attributes)
    if @user.save
      initialize_user_settings(@user)
      redirect '/validate'
    else
      redirect '/register'
    end
  else
    redirect '/register'
  end
end
```

Utilizando la técnica "*Inline Temp*", se reemplazaron variables temporales por el resultado directo. Seguidamente se observa un ejemplo de este caso:

```ruby
post '/validate' do
  if session[:code] == params[:codigo]
    user = current_user(id, session[:user_id])
    user.update_column(:valid_email, true)
  end
  redirect '/menu'
end
```

Anteriormente se contaba con una variable para almacenar el id del usuario, como se observó en ejemplos anteriores. Luego de esta refactorización, este id se lo pasa directamente por parámetro en la función `current_user`.

```ruby
def login_user(session)
  session[:user_id] = id
  Item.item_default(session)
end
```

Este método “`login_user`” se realizó para seguir simplificando el código en el controlador.

Para finalizar, se ejecutó nuevamente la herramienta Rubocop, la cual informó lo siguiente:

```
1 file inspected, 26 offenses detected, 24 offenses auto-correctable
ivan@ivan-81WA:~/Escritorio/Personal/UNRC/AyDS/AYDS-Buil-Lopez-Peruchin/Eco-Tree$
```

Al corregir las ofensas autocorregibles, se obtuvo un nuevo informe:

```
Offenses:

controllers/authentication_controller.rb:7:1: C: Style/Documentation: Missing top-level class documentation comment.
class AuthenticationController < Sinatra::Application
^^^^^
controllers/authentication_controller.rb:58:3: C: Metrics/MethodLength: Method has too many lines. [11/10]
  def check_user(user) ...
  ^^^^^^^^^^^^^^^^^^^^^^
controllers/authentication_controller.rb:74:3: C: Metrics/AbcSize: Assignment Branch Condition size for create_user is too high. [<3, 30, 5> 30.56/17]
  def create_user ...
  ^^^^^^^^^^^^^^^^
controllers/authentication_controller.rb:74:3: C: Metrics/MethodLength: Method has too many lines. [16/10]
  def create_user ...
  ^^^^^^^^^^^^^^^^
controllers/authentication_controller.rb:77:121: C: Layout/LineLength: Line is too long. [164/120]
    @user = User.create(username: params[:username], password: password_hash, email: params[:email], birthdate: params[:birthdate], leaf_id: 6, background_id: 10)
                                                                                                                                                ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

1 file inspected, 5 offenses detected, 1 offense auto-correctable
```

Esto determina que el método `create_user` resulta ser demasiado largo, por lo que se procede a refactorizar nuevamente:

Podemos observar un problema, un método determinado se encuentra con muchos parámetros, por lo que se procedió a crear un objeto con dichos parámetros, el cual será pasado como un solo parámetro a este método. Se logró esto utilizando la técnica “*Extract Variable*”.

```ruby
user_attributes = {
  username: params[:username],
  password: hash_password(params[:password]),
  email: params[:email],
  birthdate: params[:birthdate],
  leaf_id: 6,
  background_id: 10
}

@user = User.create(user_attributes)
```

A su vez, con la técnica "*Extract Method*" se realizó un nuevo método privado para comparar las dos contraseñas ingresadas cuando un nuevo usuario desea registrarse.

```ruby
# Verifica si las contraseñas coinciden
def passwords_match?
  params[:password] == params[:passwordTwo]
end
```

Por último, dentro del método `create_user` se observó una funcionalidad demasiado extensa, inicializar las configuraciones del usuario. Para resolver esto, se creó un nuevo método que encapsula estas operaciones:

```ruby
# Inicializa las configuraciones del usuario
def initialize_user_settings(user)
  session[:user_id] = user.id
  send_verificated_email(user.email, session[:code])
  PurchasedItem.create(user_id: user.id, item_id: 6)
  PurchasedItem.create(user_id: user.id, item_id: 10)
  set_item_default
end
```

Quedando finalizado el método `create_user` donde se invoca este nuevo método.

```ruby
# Método para registrar un usuario
def create_user
  if passwords_match?
    user_attributes = {
      username: params[:username],
      password: hash_password(params[:password]),
      email: params[:email],
      birthdate: params[:birthdate],
      leaf_id: 6,
      background_id: 10
    }

    @user = User.create(user_attributes)

    if @user.save
      initialize_user_settings(@user)
      redirect '/validate'
    else
      redirect '/register'
    end
  else
    redirect '/register'
  end
end
```