Tarea Ruby on Rails Avanzado (José Bustos – 20200490F)

Enunciado:

Las validaciones de modelos, al igual que las migraciones, se expresan en un mini-DSL integrado en Ruby, como muestra en el siguiente código. Escribe el código siguiente en el código dado.

```
class Movie < ActiveRecord::Base
    def self.all_ratings ; %w[G PG PG-13 R NC-17] ; end #  shortcut: array of strings
    validates :title, :presence => true
    validates :release_date, :presence => true
    validate :released_1930_or_later # uses custom validator below
    validates :rating, :inclusion => {:in => Movie.all_ratings},
        :unless => :grandfathered?
    def released_1930_or_later
        errors.add(:release_date, 'must be 1930 or later') if
        release_date && release_date < Date.parse('1 Jan 1930')
    end
    @@grandfathered_date = Date.parse('1 Nov 1968')
    def grandfathered?
        release_date && release_date < @@grandfathered_date
    end
end
```

y comprueba tus resultados en la consola:

Resultados en consola:

```
bluezelf@bluezelf:~/Documentos/Desarrollo-software-2023/Semana7/myrottenpotatoes$ rails console
Loading development environment (Rails 7.0.8)
3.0.2 :001 > m = Movie.new(:title => '', :rating => 'RG', :release_date => '1929-01-01')
 => #<Movie:0x00007f9cc0891818 id: nil, title: "", rating: "RG", description: nil, release_date:
3.0.2 :002 > m.valid?
 => false
3.0.2 :003 >
```

Vemos que da resultado falso, pero ¿porqué?, esto se debe a que la clase modelo movies, hemos puesto lineas de codigo que validan ciertos aspectos, en este caso instanciamos el objeto m a traves de la consola de rails y la asociamos a un nuevo registro en la base de datos, el cual tiene como fecha de lanzamiento (release_date => 01/01/1929), pero en el código nosotros indicamos que primero haya datos en el campo de relase_date con :presence => true, luego se valida si la fecha de lanzamiento de la película es mayor a 01/01/1930, y como vemos la nueva película tiene como fecha en 1929, por eso es que la validación es falsa.

```
3.0.2 :005 > m.errors[:title]
 => ["can't be blank"]
3.0.2 :006 > m.errors[:release_date]
 => ["must be 1930 or later"]
3.0.2 :007 > m.errors.full_messages
 => ["Title can't be blank", "Release date must be 1930 or later"]
```

Luego ponemos los comandos que nos indican, y observamos que nos da los errores previstos, el título no puede estar en blanco y que el año de lanzamiento tiene que ser mayor al de 1930.

Enunciado: Explica el siguiente código

```ruby
class MoviesController < ApplicationController
  # 'index' and 'show' methods from Section 4.4 omitted for clarity
  def new
    @movie = Movie.new
  end
  def create
    if (@movie = Movie.create(movie_params))
      redirect_to movies_path, :notice => "#{@movie.title} created."
    else
      flash[:alert] = "Movie #{@movie.title} could not be created: " +
        @movie.errors.full_messages.join(",")
      render 'new'
    end
  end
  def edit
    @movie = Movie.find params[:id]
  end
  def update
    @movie = Movie.find params[:id]
    if (@movie.update_attributes(movie_params))
      redirect_to movie_path(@movie), :notice => "#{@movie.title} updated."
    else
      flash[:alert] = "#{@movie.title} could not be updated: " +
        @movie.errors.full_messages.join(",")
      render 'edit'
    end
  end
  def destroy
    @movie = Movie.find(params[:id])
    @movie.destroy
    redirect_to movies_path, :notice => "#{@movie.title} deleted."
  end
  private
  def movie_params
    params.require(:movie)
    params[:movie].permit(:title,:rating,:release_date)
  end
end
```

new: crea una nueva instancia de la clase película, y nos redirecciona al formulario para rellenar los datos de la película.
create: crea una nueva película con los datos que fueron rellenados en el formulario, si se crea la película exitosamente te redirige a la pagina de todas las películas, y te muestra un mensaje que denota que la película ha sido creada, de lo contrario te dice que la película no ha podido crearse, con la línea “ render ‘new’ ”, nos vuelve a mostrar el formulario para rellenar los datos.
edit: nos permite editar una película existente, la busca mediante su id y le asigna la variable de instancia movie.
update: actualiza los datos puestos en el formulario de edit, pero si hay errores te muestra un mensaje y vuelve a mostrarte el formulario para editar.

destroy: elimina una película existente, la busca por su id, y redirige a la lista de las películas mostrando un mensaje de que se eliminó la película.
movie_params: esta dentro de una sección privada, la cual verifica primero que el parámetro **:movie** exista, para luego con **params[:movie].permit(:title,:rating,:release_date)**, especifique que atributos son permitidos para que se usen en la creación y edición de cada película. Esto ayuda a prevenir la asignación masiva de otros atributos no deseados que podrían ser utilizados maliciosamente.

Enunciado: Comprueba en la consola
m = Movie.create!(:title => 'STAR wars', :release_date => '27-5-1977', :rating => 'PG')
m.title # => "Star Wars"

Vemos que hace uso del metodo create!, este metodo nos indica que antes de registrar ese objeto dentro de la base de datos, el sistema hace una validación de los mismos, estas validaciones las toma gracias a que lineas arriba las hemos definido, de tal manera que si hay un error dentro de los parámetros, nos lanzaría una excepción. Esto proporciona una forma más robusta de manejar errores durante la creación del registro.

```
3.0.2 :002 > b = Movie.create!(:title => 'STAR  wars', :release_date => '27-5-1977', :rating => 'PG')
  TRANSACTION (0.0ms)  begin transaction
  Movie Create (0.2ms)  INSERT INTO "movies" ("title", "rating", "description", "release_date", "creat
ting", "PG"], ["description", nil], ["release_date", "1977-05-27 00:00:00"], ["created_at", "2023-11-1
  TRANSACTION (9.2ms)  commit transaction
 =>
#<Movie:0x00007f1a503ff5b8
...
3.0.2 :003 > b.title
 => "Star Wars"
```

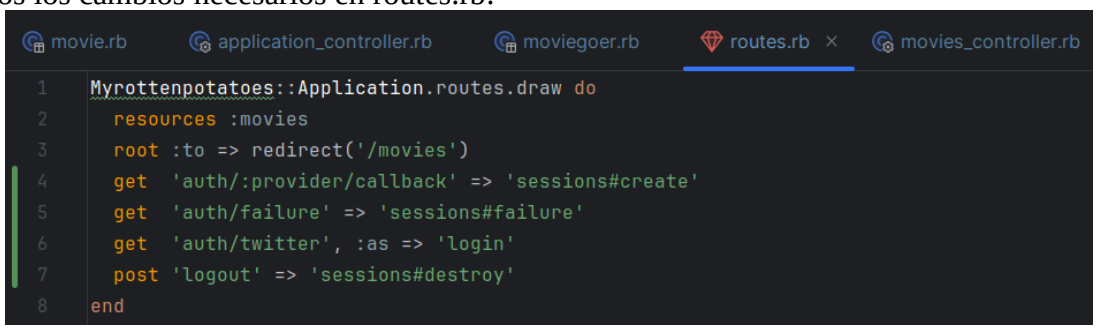
SSO y autenticación de Terceros

Generamos el modelo Moviegoer:

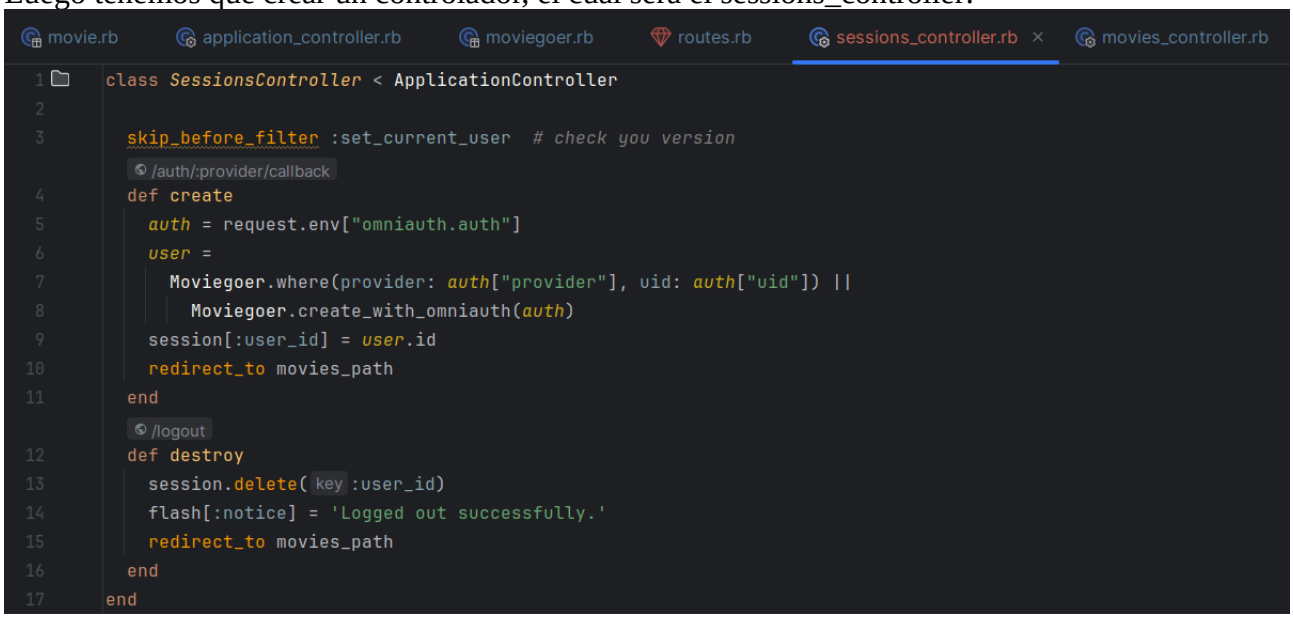
movie.rb | application_controller.rb | moviegoer.rb | movies_controller.rb

```
kapumota *
class Moviegoer < ActiveRecord::Base
    has_many :reviews
    kapumota
    def self.create_with_omniauth(auth)
        Moviegoer.create!(
        :provider => auth["provider"],
        :uid => auth["uid"],
        :name => auth["info"]["name"])
    end
end
```

Hacemos los cambios necesarios en routes.rb:



Luego tenemos que crear un controlador, el cual será el sessions_controller:



Asociaciones y Claves Foráneas

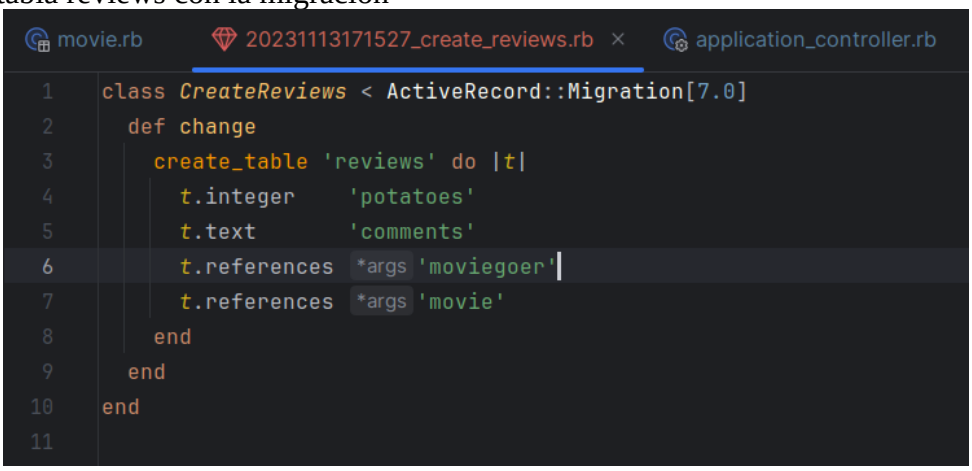
Explica la siguientes líneas de SQL:

```
SELECT reviews.*
   FROM movies JOIN reviews ON movies.id=reviews.movie_id
   WHERE movies.id = 41;
```

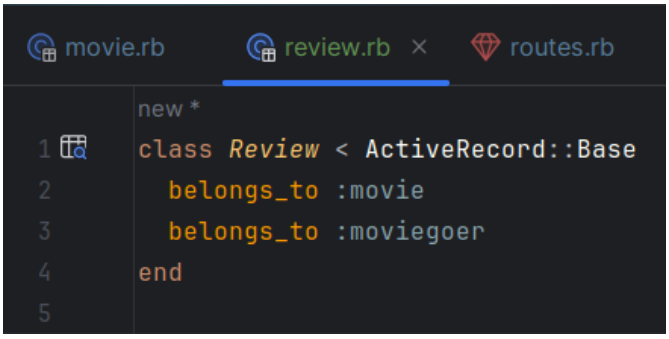
Nos indica que seleccione todas las columnas de la tabla movies, las cuales tenga su id igual al reviews.movie_id, es decir hace una asociación entre las tablas movies y reviews, una vez se relacionan las tablas busca en cuales el movie.id sea igual a 41.

Ahora nos indican que hagamos cambios dentro del código:

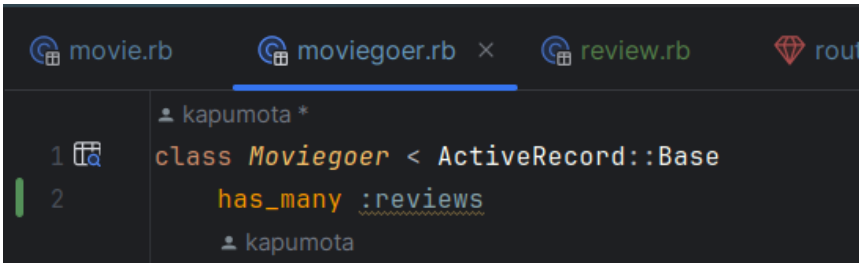
Creamos la tabla reviews con la migración



Ahora creamos un nuevo modelo review.rb



Insertamos estas lineas de codigo en movie.rb y moviegoer.rb



Enunciado:

Comprueba la implementación sencilla de asociaciones de hacer referencia directamente a objetos asociados, aunque estén almacenados en diferentes tablas de bases de datos. ¿Por que se puede hacer esto?

```
# it would be nice if we could do this:
inception = Movie.where(:title => 'Inception')
alice,bob = Moviegoer.find(alice_id, bob_id)
# alice likes Inception, bob less so
alice_review = Review.new(:potatoes => 5)
bob_review   = Review.new(:potatoes => 3)
# a movie has many reviews:
inception.reviews = [alice_review, bob_review]
# a moviegoer has many reviews:
alice.reviews << alice_review
bob.reviews << bob_review
# can we find out who wrote each review?
inception.reviews.map { |r| r.moviegoer.name } # => ['alice','bob']
```

Hay cambios sustanciales en los comandos, porque de lo contrario no funcionan:

Asociamos la pelicula “Doctor Strange in The Multiverse of Madness”, mediante si id y no mediante su nombre, ya que de lo contrario saldrá un error.

```
3.0.2 :001 > doctor = Movie.find(27)
  Movie Load (0.2ms)  SELECT "movies".* FROM "movies" WHERE "movies"."id" = ? LIMIT ?  [["id", 27], ["LIMIT", 1]]
 =>
#<Movie:0x000055fdbdc42190
...
3.0.2 :002 > doctor
 =>
#<Movie:0x000055fdbdc42190
 id: 27,
 title: "Doctor Strange in The Multiverse of Madness",
 rating: "PG-13",
 description: nil,
 release_date: Tue, 06 Apr 2021 00:00:00.000000000 UTC +00:00,
 created_at: Mon, 13 Nov 2023 16:20:30.915378000 UTC +00:00,
 updated_at: Mon, 13 Nov 2023 16:20:30.915378000 UTC +00:00>
```

A

Ahora creamos las variables poma y chan, las cuales almacenaran los registros de la tabla Moviegoer. En este caso el id de Poma es 4 y de Chandler es 5

```
3.0.2 :003 > poma,chan = Moviegoer.find(4, 5)
  Moviegoer Load (0.3ms)  SELECT "moviegoers".* FROM "moviegoers" WHERE "moviegoers"."id" IN (?, ?)  [["id", 4], ["id", 5]]
 =>
[#<Moviegoer:0x000055fdbe045648
...
3.0.2 :004 > poma
 =>
#<Moviegoer:0x000055fdbe045648
 id: 4,
 provider: "fb",
 uid: "123",
 name: "Poma",
 created_at: Wed, 15 Nov 2023 04:42:59.434178000 UTC +00:00,
 updated_at: Wed, 15 Nov 2023 04:42:59.434178000 UTC +00:00>
```

Una vez hecho esto creamos las variables reviews, poma_review y chan_review, en este comando se agrega los objetos moviegoer, ya que de lo contrario habría error.

```
3.0.2 :005 > poma_review = Review.new(:potatoes => 5, moviegoer:poma)
 => #<Review:0x000055fdbd715370 id: nil, potatoes: 5, comments: nil, moviegoer_id: 4, movie_id: nil>
3.0.2 :006 > chan_review   = Review.new(:potatoes => 3, moviegoer:chan)
 => #<Review:0x000055fdbda75ab0 id: nil, potatoes: 3, comments: nil, moviegoer_id: 5, movie_id: nil>
```

Ahora agregamos las reviews a la película la cual almacenamos en la variable doctor

```
3.0.2 :007 > doctor.reviews = [poma_review, chan_review]
  Review Load (0.4ms)  SELECT "reviews".* FROM "reviews" WHERE "reviews"."movie_id" = ?  [["movie_id", 27]]
  TRANSACTION (0.1ms)  begin transaction
  Review Create (0.8ms)  INSERT INTO "reviews" ("potatoes", "comments", "moviegoer_id", "movie_id") VALUES (?, ?, ?, ?)
"movie_id", 27]]
  Review Create (0.2ms)  INSERT INTO "reviews" ("potatoes", "comments", "moviegoer_id", "movie_id") VALUES (?, ?, ?, ?)
"movie_id", 27]]
  TRANSACTION (9.3ms)  commit transaction
 =>
[#<Review:0x000055fdbd715370 id: 1, potatoes: 5, comments: nil, moviegoer_id: 4, movie_id: 27>,
...
```

Finalmente agregamos la review de poma dentro del array de reviews, y de igualmente a chan, al final vemos quienes han hecho reviews a la película de “Doctor Strange in The Multiverse of Madness”

```
3.0.2 :008 > poma.reviews << poma_review
  Review Load (0.1ms)  SELECT "reviews".* FROM "reviews" WHERE "reviews"."moviegoer_id" = ?  [["moviegoer_id", 4]]
 => [#<Review:0x000055fdbd715370 id: 1, potatoes: 5, comments: nil, moviegoer_id: 4, movie_id: 27>]
3.0.2 :009 > chan.reviews << chan_review
  Review Load (0.3ms)  SELECT "reviews".* FROM "reviews" WHERE "reviews"."moviegoer_id" = ?  [["moviegoer_id", 5]]
 => [#<Review:0x000055fdbda75ab0 id: 2, potatoes: 3, comments: nil, moviegoer_id: 5, movie_id: 27>]
3.0.2 :010 > doctor.reviews.map{|r| r.moviegoer.name }
 => ["Poma", "Chan"]
```

Enunciado:
¿Qué indica el siguiente código SQL?

```
SELECT movies .*
   FROM movies JOIN reviews ON movies.id = reviews.movie_id
   JOIN moviegoers ON moviegoers.id = reviews.moviegoer_id
   WHERE moviegoers.id = 1;
```

Significa que muestre todas las filas las cuales la tabla movies tenga su atributo movies.id igual a reviews.id en la tabla reviews, a su vez busca que el atributo moviegoers.id tenga igual valor que reviews.moviegoer.id, en donde el valor de moviegoer.id = 1