Curso de desarrollo de software

Quinta Práctica Calificada

Indicaciones:

- Para esta evaluación utiliza solo el repositorio utilizado en tus actividades anteriores y guarda todas tus respuestas en un repositorio llamado PracticaCalificada5. **Si el estudiante utiliza el repositorio original dado en el texto , la pregunta será eliminada. La aparición de un archivo README elimina la pregunta.**
- Cada estudiante debe presentar un repositorio personal donde se encuentra esta evaluación y las otras evaluaciones dadas en clases. No se admite otros escenarios distintos. La calificación está sujeta a las actividades anteriores también.
- Las calificaciones son individuales, por lo que evita presentar solo código sin ninguna explicación o imagen. Todo ejercicio sin explicaciones o imágenes que no apoyan tus respuestas tienen nota de 0.
- No se va a utilizar alguna herramienta de apoyo como el ChatGPT por que la evaluación no lo amerita. Si de ser así, se te eliminará el examen, ya que es un ejemplo de que no has hecho las actividades ni has estudiado.

Preguntas: En este conjunto de preguntas tus respuestas deben ir de acuerdo a las actividades correspondientes, no se puntúa sino hay evidencia del uso de los scripts desarrollados y solo presentas respuestas sin evidencia de lo desarrollado a lo largo del curso. (7 puntos)

1. En las actividades relacionados a la Introducción de Rails los métodos actuales del controlador no son muy robustos: si el usuario introduce de manera manual un URI para ver (Show) una película que no existe (por ejemplo /movies/99999), verás un mensaje de excepción horrible. Modifica el método show del controlador para que, si se pide una película que no existe, el usuario sea redirigido a la vista Index con un mensaje más amigable explicando que no existe ninguna película con ese.

   Se agrega el siguiente condicional en la función def shor en el archivo movies_controller.rb para ver si la película existe o no mediante un if donde recibirá un nil en caso no exista la película:

```ruby
def show
  id = params[:id] # retrieve movie ID from URI route
  @movie = Movie.find_by(id: id) # look up movie by unique ID
  if @movie.nil?
    flash[:notice] = "La película no existe."
    redirect_to not_found_path
  else
    render(:partial => 'movie', :object => @movie) if request.xhr?
  end
  # will render app/views/movies/show.<extension> by default
end
```

Se crea además la función def not_found para renderizar la ruta:

```ruby
def not_found
  render 'not_found'
  end
```

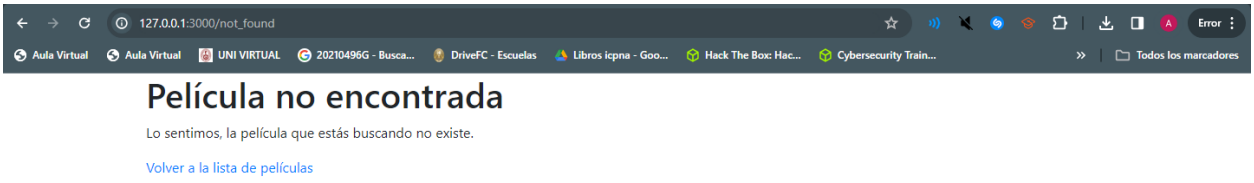
Luego en routes.rb se hace el siguiente cambio que es agegar una línea nueva que indica a donde dirigirse:

```ruby
Myrottenpotatoes::Application.routes.draw do
  mount JasmineRails::Engine => '/specs' if defined?(JasmineRails)
  #resources :movies
  resources :movies do
  resources :reviews
end
  get '/not_found', to: 'movies#not_found', as: :not_found
  root :to => redirect('/movies')
end
```

Creamos el archivo not_found.html.erb que es la vista que se verá:

```erb
<!DOCTYPE html>
<html>
<head>
  <title>Película no encontrada</title>
</head>
<body>
<h1>Película no encontrada</h1>
<p>Lo sentimos, la película que estás buscando no existe.</p>
<%= link_to "Volver a la lista de películas", movies_path %>
</body>
</html>
```

Ejecución:



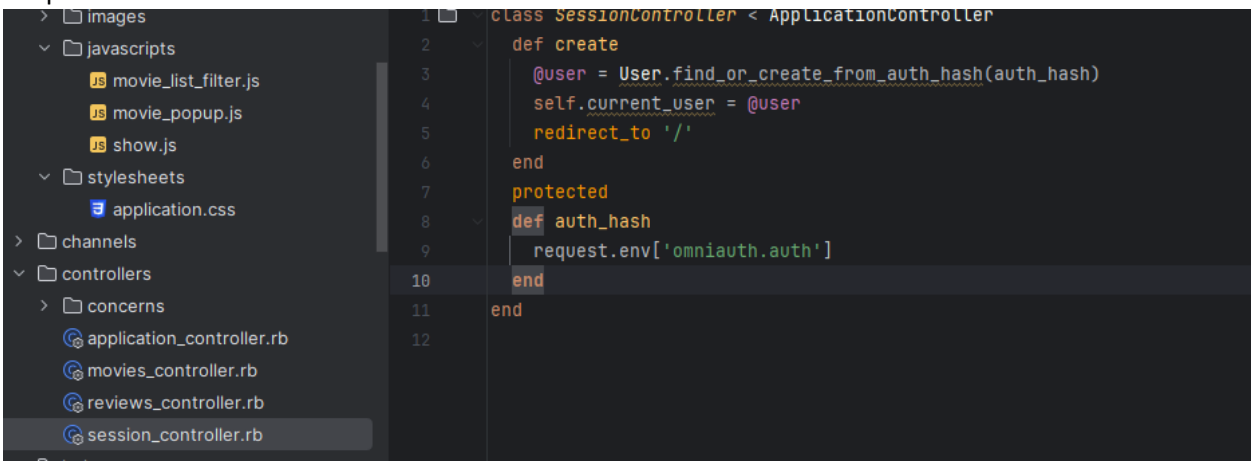
2. En las actividades relacionados a Rails Avanzado, si tenemos el siguiente ejemplo de código que muestra cómo se integra OmniAuth en una aplicación Rails:

```
class SessionsController < ApplicationController
  def create
      @user = User.find_or_create_from_auth_hash(auth_hash)
      self.current_user = @user
      redirect_to '/'
   end
 protected
 def auth_hash
         request.env['omniauth.auth']
  end
end
```

El método auth_hash tiene la sencilla tarea de devolver lo que devuelva OmniAuth como resultado de intentar autenticar a un usuario.
¿Por qué piensa que se colocó esta funcionalidad en su propio método en vez de simplemente referenciar request.env[’omniauth.auth’] directamente? Muestra el uso del script.

Esto es útil ya que si en el futuro necesitas realizar cambios en la forma en que se obtiene el hash de autenticación, solo necesitas hacer modificaciones en el método auth_hash en lugar de buscar todas las referencias a request.env['omniauth.auth'] en el código.

Script:

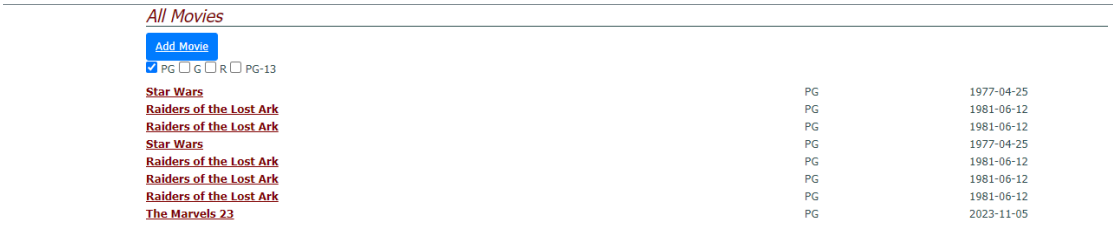


3. En las actividades relacionados a JavaScript, Siguiendo la estrategia del ejemplo de jQuery utiliza JavaScript para implementar un conjunto de casillas de verificación (checkboxes) para la página que muestra la lista de películas, una por cada calificación (G, PG, etcétera), que permitan que las películas correspondientes permanezcan en la lista cuando están marcadas. Cuando se carga la página por primera vez, deben estar marcadas todas; desmarcar alguna de ellas debe esconder las películas con la clasificación a la que haga referencia la casilla desactivada.
Para esto hacemos la siguiente modificación en show.html.rb:

```
<div id="filter-options">
  <% Movie.distinct.pluck(:rating).each do |rating| %>
    <label>
      <input type="checkbox" checked data-rating="<%= rating %>" onchange="filterMovies()">
      <%= rating %>
    </label>
  <% end %>
</div>

<div id="movies">
  <div class="row">
    <div class="col-8">Movie Title</div>
    <div class="col-2">Rating</div>
    <div class="col-2">Release Date</div>
  </div>
  <%- @movies.each do |movie| %>
    <div class="row" data-rating="<%= movie.rating %>">
      <div class="col-8"> <%= link_to movie.title, movie_path(movie) %> </div>
      <div class="col-2"> <%= movie.rating %></div>
```

```
        <div class="col-8"> <%= link_to movie.title, movie_path(movie) %> </div>
        <div class="col-2"> <%= movie.rating %></div>
        <div class="col-2"> <%= movie.release_date.strftime('%F') %> </div>
      </div>
    <% end %>
  </div>

  <script>
      const checkboxes = document.querySelectorAll('#filter-options input[type="checkbox"]');
      checkboxes.forEach(checkbox => {
          checkbox.addEventListener('change', filterMovies);
      });
      function filterMovies() {
          const movieRows = document.querySelectorAll('#movies .row');
          const selectedRatings = Array.from(checkboxes)
              .filter(checkbox => checkbox.checked)
              .map(checkbox => checkbox.getAttribute('data-rating'));
          movieRows.forEach(movieRow => {
              const rating = movieRow.getAttribute('data-rating');
              movieRow.style.display = selectedRatings.includes(rating) ? 'flex' : 'none';
          });
      }
      filterMovies();
  </script>

  </body>
  </html>
```

Ejecución:



4. De la actividad relacionada a BDD e historias de usuario crea definiciones de pasos que te permitan escribir los siguientes pasos en un escenario de RottenPotatoes:

   Given the movie "Inception" exists
   And it has 5 reviews
   And its average review score is 3.5

5. De la actividad relacionadas a BDD e historias de usuario, supongamos que en RottenPotatoes, en lugar de utilizar seleccionar la calificación y la fecha de estreno, se opta por rellenar el formulario en blanco. Primero, realiza los cambios apropiados al escenario. Enumera las definiciones de pasos a partir que Cucumber invocaría al pasar las pruebas de estos nuevos pasos. (Recuerda: rails generate cucumber:install)

6. De la actividad relacionadas a BDD e historias de usuario indica una lista de pasos como los de la siguiente figura

```
Given /I have added "(.*)" with rating "(.*)"/ do |title, rating|
    steps %Q{
        Given I am on the Create New Movie page
        When  I fill in "Title" with "#{title}"
        And   I select "#{rating}" from "Rating"
        And   I press "Save Changes"
    }
end

Then /I should see "(.*)" before "(.*)" on (.*)/ do |string1, string2, path|
    steps %Q{Given I am on #{path}}
    regexp = /#{string1}.*#{string2}/m #  /m means match across newlines
    expect(page.body).to match(regexp)
end
```

   Para implementar el siguiente paso:
   When / I delete the movie: "(.*)"/ do |title|

   Implementación:
   When /I delete the movie: "(.*)"/ do |title|
   steps %Q{
   Given I am on the Movies page
   When I follow "Delete" for "#{title}"
   Then I should not see "#{title}"
   }
   end

7. Basándose en el siguiente fichero de especificaciones (specfile), ¿a qué métodos deberían responder las instancias de F1 para pasar las pruebas?

```
require 'f1'
describe F1 do
   describe "a new f1" do
     before :each do ; @f1 = F1.new ; end
       it "should be a pain in the butt" do
       @f1.should be_a_pain_in_the_butt
     end
  it "should be awesome" do
    @f1.should be_awesome
  end
   it "should not be nil" do
    @f1.should_not  be_nil
 end
 it "should not be the empty string" do
   @f1.should_not == ""
 end
end
end
```

Deben responder a los siguientes métodos:

1. be_a_pain_in_the_butt: Debería ser un dolor en el trasero
   En este caso se espera un True

2. be_awesome: Deberia ser asombrosa
   En este caso se espera un True

3. be_nil: No debería ser nil
   En este caso se espera un False

4. ==: Indica que la cadena no debe ser vacía
   En este caso se espera un False

Pregunta: Utilizando historias de usuario y Cucumber (8 puntos)

En este ejercicio crearás historias de usuario para describir una característica de una aplicación SaaS, utilizarás la herramienta Cucumber para convertir esas historias en pruebas de aceptación ejecutables y ejecutará las pruebas en su aplicación SaaS.

Específicamente, escribirás escenarios de Cucumber que prueben los happy paths de las partes 1 a 3 de la tarea de introducción a Rails, en la que agregaste filtrado y clasificación a la vista de índice de

películas de RottenPotatoes. (Recuerda, los happy paths son los pasos que siguen los usuarios cuando utilizan una aplicación con éxito).
El código de la aplicación en rottenpotatoes contiene una solución "canónica" para la tarea de introducción de Rails sobre la cual escribir sus escenarios y el andamiaje necesario para los primeros escenarios. Utiliza la carpeta comprimida dada en la actividad.

Ahora que tienes el código, deberías poner en funcionamiento RottenPotatoes. Este es el mismo conjunto de pasos que damos en la mayoría de las aplicaciones Rails nuevas.

```
cd rottenpotatoes
bundle install --without production
bin/rake db:setup
bin/rails server -b 0.0.0.0
```

Verifica que puedas cargar RottenPotatoes y que tenga algunas películas en la base de datos.

Recuerda que los pasos dados de una historia de usuario especifican el estado inicial del sistema: no importa cómo llegó el sistema a ese estado.
El objetivo de BDD es expresar tareas conductuales en lugar de operaciones de bajo nivel.

Antes de continuar verifica que tu aplicación esté configurada correctamente y sin errores:

```
bundle exec cucumber
```

Lee el resultado. ¿Tiene sentido? (Se esperan pruebas fallidas porque aún no las ha implementado).

Ejemplo de producción de Cucumber

Para toda esta tarea, debes modificar movie_steps.rb en features/step_definition así como sort_movie_list.feature , filter_movie_list.feature en features/

Se creará una definición de paso que coincidirá con el paso `Given the following movies exist` en la sección Background de sort_movie_list.feature y filter_movie_list.feature.

Agrega tu código en el archivo de definición de pasos movie_steps.rb. Puedes utilizar llamadas ActiveRecord para agregar películas directamente a la base de datos, está bien omitir la GUI asociada con la creación de nuevas películas, ya que eso no es lo que se prueban en estos escenarios.

Elimina cualquier código pendiente (pending) después de implementar la definición de cada paso. Todo funciona bien cuando todos los pasos en `Background` para los escenarios en filter_movie_list.feature y sort_movie_list.feature pasan a Verde. **(1 punto)**

Resuelve cada una de las preguntas:

1. Completa el escenario **restrict to movies with PG or R ratings in filter_movie_list.feature.** Puedes utilizar las definiciones de pasos existentes en **web_steps.rb** para marcar y desmarcar las casillas correspondientes, enviar el formulario y comprobar si aparecen las películas correctas (y, lo que es igualmente importante, no aparecen las películas con clasificaciones no seleccionadas).
2. Dado que es tedioso repetir pasos como **When I check the "PG" checkbox, And I check the "R" checkbox**, etc., crea una definición de paso que coincida con un paso como, por ejemplo: **Given I check the following ratings: G, PG, R**
   Esta definición de un solo paso solo debe marcar las casillas especificadas y dejar las demás casillas como estaban.
3. Dado que es tedioso especificar un paso para cada película individual que deberíamos ver, agrega una definición de paso para que coincida con un paso como: "**Then I should see the following movies"**.
4. Para el escenario **all ratings selected** sería tedioso utilizar **And I should see** para nombrar cada una de las películas. Eso restaría valor al objetivo de BDD de transmitir la intención de comportamiento de la historia del usuario. Para solucionar este problema, completa la definición de paso que coincida con los pasos del formulario: Then I should see all the movies en movie_steps.rb.
   Considera contar el número de filas en la tabla HTML para implementar estos pasos. Si ha calculado las filas como el número de filas de la tabla, puede usar la afirmación expect(rows).to eq value para fallar la prueba en caso de que los valores no coincidan.

5. Utiliza tus nuevas definiciones de pasos para completar el escenario con todas las calificaciones seleccionadas. Todo funciona bien si todos los escenarios en filter_movie_list.feature pasan con todos los pasos en verde.

Observación debes responder todas las preguntas (1-5) para que puntué, sino será 0 la calificación (5 puntos)

6. Dado que los escenarios en sort_movie_list.feature implican clasificación, necesitarás la capacidad de tener pasos que prueben si una película aparece antes que otra en la lista de salida. Cree una definición de paso que coincida con un paso como: Then I should see "Aladdin" before "Amelie". Utiliza:
   a. page es el método Capybara que devuelve un objeto que representa la página devuelta por el servidor de aplicaciones. Puedes usarlo en expectativas como expect(page).to have_content('Hello World'). Más importante aún, puedes buscar en la página elementos específicos que coincidan con selectores CSS o expresiones XPath. Consulta la documentación querying de Capybara: https://github.com/teamcapybara/capybara
   b. page.body es el cuerpo HTML de la página como una cadena gigante.
   c. Una expresión regular podría capturar si una cadena aparece antes que otra en una cadena más grande, aunque esa no es la única estrategia posible **(1 punto).**
7. Utiliza la definición de paso que creaste anteriormente para completar los escenarios, ordenar películas alfabéticamente y ordenar películas en orden creciente según la fecha de lanzamiento en sort_movie_list.feature. Todo funciona bien son todos los pasos de todos los escenarios en ambos archivos de funcionalidades pasan a verde. **(1 punto)**

Asegúrate de que tu código esté dentro de este proyecto en la siguiente ubicación:
rottenpotatoes/features y muestra el resultado de éxito.

Pregunta (5 puntos) : Para el siguiente ejercicio utiliza la lista de proyectos Rails de código abierto en Open Source Rails: https://github.com/gramantin/awesome-rails#open-source-rails-apps

Analizados:

- activeWorkflow : una plataforma inteligente de automatización de procesos y flujos de trabajo basada en agentes de software (que utilizan Rails 6.0).
- airCasting : una plataforma para registrar, mapear y compartir datos ambientales y de salud usando su teléfono inteligente (usando Rails 6.1).
- alonetone : una aplicación de alojamiento, gestión y distribución de música (que utiliza Rails 7.0)
- asakusaSatellite : una aplicación de chat en tiempo real para desarrolladores (que utiliza Rails 6.0).

1. Describa uno o más patrones de diseño que podrían ser aplicados al diseño del sistema.
   Patrón de Diseño Observer:
   Este patrón podría ser aplicado en sistemas como activeWorkflow o airCasting.
   El patrón Observer se utiliza para definir una dependencia de uno a muchos entre objetos para que cuando uno de los objetos cambie de estado, todos sus dependientes sean notificados y actualizados automáticamente.
   En activeWorkflow, donde hay procesos y flujos de trabajo, puede haber varios elementos que necesiten reaccionar a cambios en el estado de otros elementos.

   Patrón de Diseño MVC (Model-View-Controller):
   El patrón MVC podría ser aplicado en sistemas como alonetone o asakusaSatellite. MVC separa la lógica de negocios (Model), la presentación (View) y la gestión de eventos y la entrada del usuario (Controller). En aplicaciones como alonetone, donde hay manipulación de datos y representación visual de la música, y en aplicaciones de chat como asakusaSatellite, donde la entrada del usuario y la representación visual deben estar separadas de la lógica subyacente, MVC es un patrón de diseño eficaz.

2. Dado un sistema simple que responde a una historia de usuario concreta, analice y elija un paradigma de diseño adecuado

   Historia de Usuario: Usuario quiere poder realizar búsquedas avanzadas en la plataforma airCasting para encontrar datos específicos de calidad del aire.

   Paradigma de Diseño: El paradigma de diseño adecuado para esta historia de usuario podría ser el Paradigma de Diseño de Filtros o Pipes. Este paradigma se centra en la construcción de sistemas modulares y reutilizables, donde cada módulo realiza una función específica y los datos fluyen a través de una serie de transformaciones. En el caso de airCasting, se pueden tener

filtros específicos para datos de calidad del aire, y los usuarios podrían combinar estos filtros para realizar búsquedas avanzadas.

3. Analice y elija una arquitectura software apropiada que se ajuste a una historia de usuario concreta de este sistema. ¿La implementación en el sistema de esa historia de usuario refleja su idea de arquitectura?

   Historia de Usuario: Usuario quiere recibir notificaciones automáticas en activeWorkflow cuando un proceso específico ha sido completado con éxito.

   Arquitectura Software: La Arquitectura basada en Microservicios podría ser apropiada. En esta arquitectura, cada función o proceso se implementa como un servicio independiente, y estos servicios pueden comunicarse entre sí. Para la notificación automática, se puede implementar un servicio de notificaciones que sea independiente del servicio de procesos. Cuando un proceso se completa con éxito, el servicio de procesos puede notificar al servicio de notificaciones, que luego se encargará de enviar la notificación al usuario.

   En activeWorkflow, esta arquitectura permitiría escalabilidad y mantenimiento independiente de cada componente, lo que facilita la incorporación de nuevas características y mejora la robustez del sistema. La implementación de esta historia de usuario podría implicar la introducción de un nuevo servicio de notificaciones que se comunique con el servicio de procesos.