Curso de desarrollo de software

Examen Final

Normas:

1. No compartir respuestas/consultas con tus compañeros a través de chats, redes sociales u otros medios digitales. Se va a eliminar la evaluación de manera automática si un estudiante es sorprendido.
2. Todo acto anti-ético será amonestado y registrado en el historial del estudiante.
3. Pese a lo útil que es la Inteligencia Artificial con el chatGPT o herramientas similares (repositorios relacionados al curso) no se permite el uso de herramientas IA en el examen.
4. Presenta imágenes de tus respuestas junto con el código desarrollado. Tú código debe ser probado sino no puntúa. Todo examen que solo presenta código sin nada que lo soporte o respuestas sin explicación o puntuales será no considerada en la calificación. Mira un ejemplo:

   Pregunta: ¿Las pruebas también sirven como parte de la documentación?
   Respuesta de un estudiante: SI si respondes así , por que las pruebas sirven para la documentación como vimos en clase la nota es 0

Ejemplo de respuestas: Si las pruebas también pueden servir como documentación para las funciones que se implementa en una aplicación. Otros desarrolladores del equipo pueden leer tus pruebas y descubrir fácilmente qué busca lograr tu código.

Al crear las pruebas primero, puede asegurarse de que la función que está probando funcione de la forma prevista y evitar sorpresas en el futuro.

Sube tus respuestas a un repositorio llamado ExamenFinal-CC3S2 y dentro escribe las carpetas de cada una de las partes (preguntas) del examen. Escribe un archivo .md para explicar tus respuestas.

Parte 1

Estas preguntas corresponden a las actividades desarrolladas en clase.

1. Produce un conflicto de fusión (merge) en algún repositorio de tus actividades realizadas. Establece los pasos y comandos que usas para resolver un conflicto de fusión en Git. Si intentas git push y falla con un mensaje como : *Non-fast-forward (error): failed to push some refs* esto significa que algún archivo contiene un conflicto de fusión entre la versión de tu repositorio y la versión del repositorio origen. Para este ejercicio debes presentar el conflicto dado, los pasos y comandos para resolver el problema y las solución.
2. Digamos que nos dan el modelo de User de la siguiente manera:

class User < ActiveRecord::Base

validates :username, :presence => true

validate :username_format

end

Revisa la documentación sobre validaciones si fuese necesario: https://guides.rubyonrails.org/active_record_validations.html . El primero validates actua sobre la campo username. Un objeto User no será válido sin un atributo username.

La palabra clave validate funciona de manera diferente a validates. validate toma un método/bloque y lo utiliza para validar registros cuando se modifican o se insertan en la base de datos. Revisa: https://api.rubyonrails.org/v5.1.7/classes/ActiveModel/Validations/ClassMethods.html#method-i-validate

1. ¿Qué pasa si tenemos @user sin nombre de usuario y llamamos a @user.valid? ¿Qué guardará @user.save

2. Implementa username_format. Para los propósitos, un nombre de usuario comienza con una letra y tiene como máximo 10 caracteres de largo. Recuerda, las validaciones personalizadas agregan un mensaje a la colección de errores.

3. Recuerda, los filtros nos ayudan a verificar si ciertas condiciones se cumplen antes de permitir que se ejecute una acción del controlador. Para el modelo de User, digamos que queremos verificar si @user era administrador de todos los métodos en AdminController. Completa el método before_filter:check_admin a continuación que verifica si el campo de administrador en @user es verdadero. De lo contrario, redirija a la página admin_login con un mensaje que indica acceso restringido.

```
class AdminController < ApplicationController

        before_filter :check_admin

    # Completa el codigo
```

4. AJAX (JavaScript y XML asíncronos) es un grupo de herramientas y técnicas para el desarrollo de aplicaciones web asíncronas. El objetivo de AJAX es que la comunicación entre una aplicación y el servidor de datos (es decir, solicitudes HTTP) no interfiera con la experiencia, la visualización y el comportamiento de la aplicación. A continuación, se te proporciona un formulario que simula el inicio de sesión. Comprueba si la combinación de nombre de usuario y contraseña funciona junto con la cuenta, si la hay. Para hacer eso, queremos que se realice una solicitud HTTP POST cuando se envíe este formulario. Escribe tu solución con jQuery y comenta dónde debe ubicarse la función de devolución de llamada (callback). Comprueba tus resultados.

```
<form method="POST" id="foo">

        <input type="text" class="user" />

        <input type="password" class="pass" />

        <input type="button" value="Log in" id="onSubmit" />
```

```
</form>

$("#onSubmit").click(function() {

# Tu codigo

})
```

5. ¿Cuándo deberías utilizar la metaprogramación basada en eval en lugar de la metaprogramación basada en bloques?.

Parte 2. Pruebas

Observación en cada una de los pasos del ejercicio debes responder cada uno de los ítems para cada se puntúe en orden.

En este ejercicio usarás TDD para desarrollar la funcionalidad RottenPotatoes que permite al usuario buscar una película en TMDb. Específicamente, usarás TDD para crear un controlador, que recibe la solicitud del usuario, y un modelo que en realidad llama al servicio TMDb remoto para obtener información sobre la película especificada.

Utiliza la carpeta comprimida dado en la evaluación.

Asegúrate de ejecutar bundle install --without para configurar todas las dependencias. Aquí trabajarás con TMDb API y Guard para automatizar el flujo de trabajo (puedes utilizar algunas otras si crees necesario) . También utilizarás Faraday, unalibreríacliente HTTP, para realizar llamadas API. Edita el Gemfile para incluir las siguientes gemas:

```
gem 'faraday'
group :test do
  gem 'rails-controller-testing'
  gem 'guard-rspec'
end
```

Realiza lo siguiente:

- Vuelva a ejecutar bundle install para obtener las gemas. Luego ejecuta Rails generate rspec:install para asegurarte de que los archivos que RSpec que necesitas estén en su lugar.
- Edita el archivo spec/rails_helper.rb para incluir require 'byebug' en la parte superior, de modo que puedas acceder al depurador según sea necesario para que las pruebas funcionen.
- Ejecuta el paquete exec guard init rspec para configurar los archivos necesarios para Guard, lo que dará como resultado la creación de un nuevo Guardfile. Agrega ese archivo a tu repositorio.
- Configura la base de datos con el comando habitual
- Ejecuta el servidor para mostrar que todo este bien.

Paso 1: Escribiendo una nueva vista (2 puntos)

Como primer paso, ampliarás RottenPotatoes con un formulario que permita al usuario buscar en The Open Movie Database (TMDb) una película para agregar a RottenPotatoes.

Recuerda que en la arquitectura MVC, el controlador es el "primer punto de entrada" a la aplicación cuando el cliente realiza una solicitud, por lo que tendrás que elegir un nombre para la acción del controlador que eventualmente manejará el envío de este formulario. Llamarás a la acción del controlador search_tmdb. Lo primero que harás será crear la vista correspondiente a esa acción del controlador. Esta vista eventualmente se mostrará cuando un usuario navegue a /search_tmdb. Haz incluido un archivo de inicio en search_tmdb.html.erb pero actualmente le faltan dos cosas. Primero los parámetros del formulario con los que se debe enviar y segundo un botón que nos lleva de regreso a la página de inicio.

Incluya la ruta y el método (¿qué tipo de solicitud estás realizando?) en form_tag proporcionado en la vista. Utiliza search_tmdb_path para la ruta. Además, incluya un campo de identificación para la etiqueta del formulario con tmdb_form.

Supuestamente puedes buscar películas en TMDb. Pero no hay forma de que puedas ir y regresar a la página de inicio sin cambiar manualmente el URI. Agrega un botón que llevará al usuario a la página de búsqueda y agrega un botón para llevarlo de regreso a la página de inicio. Observa cómo los usuarios regresan a la vista existente show.html.erb para inspirarse. ¿Tienes todas las piezas necesarias para ir a /search_tmdb ahora?

En la arquitectura MVC, el trabajo del controlador es responder a la interacción del usuario, llamar a los métodos del modelo apropiados para recuperar o manipular los datos necesarios y generar una vista adecuada. Por lo tanto, puedes describir el comportamiento deseado de nuestro método de controlador, aún inexistente, de la siguiente manera:

1. Esperar que el método controlador llame a un método modelo para realizar la búsqueda en TMDb, pasándole los términos de búsqueda escritos por el usuario.
2. Luego, esperar que el método del controlador seleccione la plantilla de vista correcta (Show Search Results) para renderizar.
3. Finalmente, esperar que los resultados de búsqueda reales estén disponibles para esa plantilla.

Ten en cuenta que ninguno de los métodos o plantillas de esta lista existe todavía. Esa es la esencia de TDD: escribir una lista concreta y concisa de los comportamientos deseados (la especificación -spec) y utilizarla para impulsar la creación de métodos y plantillas.

Entonces, crea el archivo spec/controllers/movies_controller_spec.rb que contenga lo siguiente, que corresponde a las tres expectativas articuladas anteriormente. (Por convención sobre la configuración, RSpec espera que las especificaciones de app/controllers/movies_controller.rb residan en spec/controllers/movies_controller_spec.rb; de manera similar para los modelos).

```
require 'rails_helper'


describe MoviesController do
  describe 'searching TMDb' do
    it 'calls the model method that performs TMDb search'
    it 'selects the Search Results template for rendering'
    it 'makes the TMDb search results available to that template'
```

```
  end
end
```

La línea 3 dice que las siguientes especificaciones **describe** el comportamiento de la clase MoviesController. Debido a que esta clase tiene varios métodos, la línea 4 dice que este primer conjunto de especificaciones describe el comportamiento del método que busca en TMDb. Como puedes ver, describe puede ir seguido de un nombre de clase o una cadena de documentación descriptiva, y los bloques de descripción pueden anidarse.

Las siguientes tres líneas son marcadores de posición para lo que RSpec llama examples, un breve fragmento de código que prueba un comportamiento específico. Aún no has escrito ningún código de prueba, por lo que puedes ejecutar estos esqueletos de prueba.

Hay tres formas de hacerlo (pruébalas ahora):

1. La ejecución de bundle exec rspec *archivo* ejecuta las pruebas en un solo archivo, como movies_controller_spec.rb
2. Al ejecutar bundle exec rspec sin argumentos se ejecutan todas las pruebas.
3. Ejecutar bundle exec guard. Una vez que Guard se esté ejecutando, observarás los cambios en tu código o en tus archivos RSpec y volver a ejecutar automáticamente las especificaciones que creas que podrían verse afectadas.

Para el resto del ejercicio, se asume que cuando cambia las pruebas o el código, se ejecuta guard y obtiene comentarios inmediatos, o vuelves a ejecutar manualmente las especificaciones necesarias.

En cualquier caso, puede ver que los ejemplos (cláusulas it) que no contienen código se muestran en amarillo como "pendientes".

Como sabes, en una aplicación Rails el hash de parámetros se completa automáticamente con los datos enviados en un formulario para que el método del controlador pueda examinarlos. Afortunadamente, RSpec proporciona un método get que simula el envío de un formulario a una acción del controlador: el primer argumento es el nombre de la acción (método del controlador) que recibirá el envío y el segundo argumento es un hash que se convertirá en los params vistos por la acción del controlador. Ahora puedes escribir la primera línea de la primera especificación, pero debes superar un par de obstáculos para llegar a la fase Roja de Red-Green-Refactor.

Observación: Si ve errores con una palabra clave ThreadError, incluye el siguiente fragmento de código en la parte superior de movies_controller_spec.rb.

```
if RUBY_VERSION>='2.6.0'
  if Rails.version < '5'
    class ActionController::TestResponse < ActionDispatch::TestResponse
      def recycle!
        # hack to avoid MonitorMixin double-initialize error:
        @mon_mutex_owner_object_id = nil
        @mon_mutex = nil
        initialize
      end
```

```
   end
  else
    puts "Monkeypatch for ActionController::TestResponse no longer needed"
  end
end
```

Paso 2: Lograr que se apruebe la primera especificación (2 puntos)

Mientras que un it sirve como marcador de posición para un ejemplo aún por escribir, un bloque do...end es un caso de prueba real. Modifica el primer ejemplo para que se vea así:

```
it 'calls the model method that performs TMDb search' do
  get :search_tmdb, {:search_terms => 'hardware'}
end
```

Este ejemplo muestra que RSpec for Rails incluye un método get que simula realizar un HTTP GET en tu aplicación. El argumento hash representa el contenido exacto de params[] que tu aplicación vería, por lo que si estás utilizando un asistente de formulario (como form_tag_helper) para construir los formularios, necesitas saber cuáles serán los nombres de los campos HTML para poder pasarlos. los valores correctos en este hash. Cuando ejecutes esta especificación (o guard la ejecute por ti), fallará estrepitosamente, porque no hay ninguna ruta que coincida con GET /search_tmdb (e incluso si hubiera una ruta, no habría ninguna acción del controlador definida para recibirla).

Antes de continuar, asegúrese de que la especificación anterior esté en verde. Muestra eso en la evaluación.

Dicho de otra manera, esa línea de código de prueba te lleva a asegurarte de que el nuevo método de controlador y la vista que finalmente representará tengan los nombres correctos y una ruta coincidente. ¿Puedes ir a /search_tmdb ahora?

En este punto, RSpec debería informa Green para el primer ejemplo, pero eso no es realmente exacto porque el ejemplo en sí está incompleto: en realidad no se ha verificado si search_tmdb en el controlador llama a un método modelo para buscar TMDb, como lo requiere la especificación.

Esta prueba no trata de si el método del modelo (aún no existente) hace lo correcto. Todo lo que estás verificando aquí es que el controlador intenta llamar a ese método y le pase los argumentos correctos (en este caso, el valor que el usuario escribió en el campo del formulario).

Entonces, esta prueba no se trata del método del modelo, ¡sino que la acción del controlador que estás a punto de probar debe llamar al método del modelo!

El método modelo es el código que deseas tener (¿qué tal 'mwh' para abreviar?). Como no lo tienes, configura un doble para el método del modelo y verifica que se llame al doble.

El mvh tendrá que ser un método de clase, ya que buscar películas en TMDb es un comportamiento relacionado con películas en general y no con una instancia particular de una película. Es de suponer que

el mvh aceptaría una cadena que representa lo que el usuario escribió y devolvería una colección de instancias coincidentes como objetos Movie. ¿Qué está pasando aquí?.

De todos modos, si el mvh anterior existiera el método controlador lo llamaría así:

@movies = Movie.find_in_tmdb(params[:search_terms])

Modifica la prueba de controlador para verificar que la acción del controlador llame a dicho método:

```
it 'calls the model method that performs TMDb search' do
  fake_results = [double('movie1'), double('movie2')]
  expect(Movie).to receive(:find_in_tmdb).with('hardware').
    and_return(fake_results)
  get :search_tmdb, {:search_terms => 'hardware'}
end
```

Dado que find_in_tmdb es un código que aún no tienes, el objetivo aquí es "falsificar" el comportamiento que mostraría si lo tuvieses. En particular, se utiliza el método double de RSpec para crear un arreglo de dos objetos Movie "dobles de prueba". Mientras que un objeto Movie real respondería a métodos como title y rating, el doble de prueba generaría una excepción si invocara algún método en él. Dado este hecho, ¿por qué usarías dobles? El motivo es aislar estas especificaciones del comportamiento de la clase Movie, que podría tener tus propios errores. Los objetos mocks, o simplemente mocks, son como marionetas cuyo comportamiento controlas completamente, lo te permite aislar las pruebas unitarias de tus clases colaboradoras y mantener las pruebas independientes (la I en FIRST).

De hecho, en RSpec, un alias para double es mock.

¿Qué sucede cuando ejecutas esta especificación?

Desafortunadamente, no puedes simplemente fingir que el método find_in_tmdb se agregará en tiempo de ejecución, tendrías que indicarlo explícitamente en el modelo Movie. Agrega las siguientes líneas a movie.rb.

```
def self.find_in_tmdb(search_terms)
end
```

El uso de expect... to receive para reemplazar temporalmente un método "real" con fines de prueba es un ejemplo del uso de un seam.

La siguiente línea (en realidad una continuación de lo esperado) especifica que find_in_tmdb debería devolver la colección de dobles que configuras en la línea 6. Esto completa la ilusión del "código que deseas tener": estas llamando a un método que no no existe todavia y proporciona los resultados que deseas si existiera!. Si omites with, RSpec aún verificará que se llame a find_in_tmdb, pero no verifica si los argumentos son los que esperabas. Si omites and_return, la llamada al método fake devolverá nil en lugar de un valor elegido por nosotros.

En cualquier caso, después de ejecutar cada ejemplo, RSpec realiza un paso de desmontaje que restaura las clases a su condición original, por lo que si quisieras realizar estas mismas falsificaciones en otros ejemplos, necesitarias especificarlas en cada uno de ellos. Este desmontaje automático es otra parte importante para mantener las pruebas independientes.

Ahora la prueba falla, pero falla por la razón correcta: se espera que la acción del controlador search_tmdb llamara a un método llamado find_in_tmdb en la clase Movie, pero no lo hizo. Ahora puedes modificar el código en MoviesController#search_tmdb para que pase la prueba:

@movies = Movie.find_in_tmdb(params[:search_terms])

¿Por qué la expectativa receive debe preceder a la acción get en la prueba?. Muestra código aquí de que sucecería por ejemplo de la acción get en tu respuesta.

Paso 3: Más comportamientos de controlador (3 puntos)

Volviendo al esqueleto de archivo de especificaciones original, la siguiente especificación dice que search_tmdb debe seleccionar la vista "Search Results" para renderizar.

Dado que el comportamiento predeterminado de Rails es intentar representar una vista cuyo nombre y ruta coincidan con la acción del controlador, en este caso app/views/movies/search_tmdb.html.erb (o .html.haml si lo prefieres), la especificación solo necesita para verificar que la acción del controlador realmente intentará representar esa plantilla de vista. Para hacer esto usaremos el método reponse de RSpec-Rails: una vez que hayamos realizado un get o post en una especificación del controlador, el objeto devuelto por el método de response contendrá la respuesta del servidor de aplicaciones a esa acción, y podemos aseverar una expectativa que la respuesta habría dado una visión particular. Al igual que el método get utilizado para desarrollar la especificación del controlador, el método response es una función proporcionada por la gema rspec-rails, no por RSpec en sí.

En el siguiente código que puede conectar al ejemplo correcto, la condición de coincidencia para la expectativa la proporciona render_template(), por lo que la aseveración se cumple si el objeto (en este caso, la respuesta de la acción del controlador) intentó representar un determinado vista. La aseveración negativa expect(...) to_not (o su alias, expect(...).not_to) se puede utilizar para especificar que la condición de coincidencia no debe cumplirse.

```
it 'selects the Search Results template for rendering' do
  fake_results = [double('movie1'), double('movie2')]
  allow(Movie).to receive(:find_in_tmdb).and_return(fake_results)
  get :search_tmdb, {:search_terms => 'hardware'}
  expect(response).to render_template('search_tmdb')
end
```

¿Qué sucede aquí?. Explica el código y realiza algunos cambios para demostrar tus resultados. La respuesta debe ser completa, hay muchos detalles a analizar.

Antes de escribir otro ejemplo, consideremos el paso Refactor de Red-Green-Refactor. Modifica el archivo de especificaciones de tu controlador para que se vea así:

```
require 'rails_helper'

describe MoviesController do
  describe 'searching TMDb' do
    before :each do
      @fake_results = [double('movie1'), double('movie2')]
    end
    it 'calls the model method that performs TMDb search' do
      expect(Movie).to receive(:find_in_tmdb).with('hardware').
        and_return(@fake_results)
      get :search_tmdb, {:search_terms => 'hardware'}
    end
    it 'selects the Search Results template for rendering' do
      allow(Movie).to receive(:find_in_tmdb).and_return(@fake_results)
      get :search_tmdb, {:search_terms => 'hardware'}
      expect(response).to render_template('search_tmdb')
    end
    it 'makes the TMDb search results available to that template'
  end
end
```

Como lo implica el nuevo código anterior, ahora hay un bloque de configuración que se ejecuta antes (before) each cada uno de los ejemplos dentro del grupo de describe, similar a la sección Background de una característica de Cucumber, cuyos pasos se realizan antes de cada escenario. (También existe before(:all), que ejecuta el código de configuración solo una vez para un grupo completo de pruebas, pero corre el riesgo de que tus pruebas dependan unas de otras al usarlo, ya que es fácil que se introduzcan dependencias difíciles de depurar. que solo se exponen cuando las pruebas se ejecutan en un orden diferente o cuando solo se ejecuta un subconjunto de pruebas).

Si bien el concepto de factorizar la configuración común en un bloque before es sencillo, se tuvo que realizar un cambio sintáctico para que funcionara, debido a la forma en que se implementa RSpec. Específicamente, se tuvo que cambiar fake_results por una variable de instancia @fake_results, porque las variables locales que ocurren dentro de cada ejemplo no son visibles fuera de ese ejemplo. Por el contrario, las variables de instancia de un grupo de ejemplo son visibles para todos los ejemplos de ese grupo. Dado que estás configurando el valor en el bloque before: each, cada caso de prueba verá el mismo valor inicial de @fake_results.

¿De qué tipo de objeto crees que @fake_results es una variable de instancia? (Dicho de otra manera, ¿cuál crees que es el valor de self dentro de un bloque de código de prueba?)

Para verificar que el controlador asigna las variables de vista correctas ,esto se comienza con una lista de tres descripciones de comportamientos deseados en la acción de controlador:

```
describe 'searching TMDb' do
  it 'calls the model method that performs TMDb search'
  it 'selects the Search Results template for rendering'
```

```
  it 'makes the TMDb search results available to that template'
end
```

Sólo queda un ejemplo por escribir para comprobar que los resultados de la búsqueda de TMDb estarán disponibles en la vista de respuesta.

El método RSpec assigns() realiza un seguimiento de qué variables de instancia se asignaron en el método del controlador. Por lo tanto, assigns(:movies) devuelve cualquier valor (si corresponde) asignado a @movies por la acción del controlador, y la especificación solo tiene que verificar que la acción del controlador configura correctamente esta variable.

En el caso, ya hemos acordado devolver los dobles como resultado de la llamada al método falsificado, por lo que el comportamiento correcto para search_tmdb sería establecer @movies en este valor. Al hacerlo y factorizar el código común entre la segunda y tercera especificación en tu propio describe anidada, obtenemos esta versión final de la especificación del controlador. Elegimos la cadena de descripción " after valid search " para nombrar este bloque describe anidado porque todos los ejemplos de este subgrupo suponen que se ha producido una llamada válida a find_in_tmdb. Esa suposición en sí misma se prueba con el primer ejemplo (prueba eso). Cuando se anidan grupos de ejemplo, los bloques before asociados con el anidamiento externo se ejecutan antes que los asociados con el anidamiento interno.

```
 1 require 'rails_helper'
 2
 3 describe MoviesController do
 4   describe 'searching TMDb' do
 5     before :each do
 6       @fake_results = [double('movie1'), double('movie2')]
 7     end
 8     it 'calls the model method that performs TMDb search' do
 9       expect(Movie).to receive(:find_in_tmdb).with('hardware').
10         and_return(@fake_results)
11       get :search_tmdb, {:search_terms => 'hardware'}
12     end
13     describe 'after valid search' do
14       before :each do
15         allow(Movie).to receive(:find_in_tmdb).and_return(@fake_results)
16         get :search_tmdb, {:search_terms => 'hardware'}
17       end
18       it 'selects the Search Results template for rendering' do
19         expect(response).to render_template('search_tmdb')
20       end
21       it 'makes the TMDb search results available to that template' do
22         expect(assigns(:movies)).to eq(@fake_results)
23       end
24     end
```

```
25   end
26 end
```

Ejecuta este código y explica las líneas 5-7, 14-17, 18-20. ¿Hacemos stubbing?.

Especifica cuales son las construccionesde RSpec se utiliza para (a) crear un seam, (b) determinar el comportamiento de un seam.

¿Por qué suele ser preferible utilizar before(:each) en lugar de before(:all)?. Explica un caso de ejemplo.

Paso 4: TDD para el modelo (3 puntos)

A continuación usarás TDD para implementar el método del modelo find_in_tmdb que has estado eliminando hasta ahora. Dado que se supone que este método llama al servicio TMDb real, nuevamente necesitas usar stubping, esta vez para evitar que los ejemplos dependan del comportamiento de un servicio de Internet remoto.

Como muchas aplicaciones SaaS, TMDb tiene una API RESTful que admite acciones como "buscar películas que coincidan con una palabra clave" o "recuperar información detallada sobre una película específica". Para evitar abusos y rastrear a cada usuario de la API por separado, cada desarrollador que desee realizar llamadas a la API debe primero obtener su propia clave API solicitándola a través del sitio web de TMDb. Las solicitudes de API que no incluyen una clave de API válida no se aceptan y, en su lugar, se devuelve un error. Para las URI de solicitud que contienen una clave API válida, TMDb devuelve un objeto JSON como resultado de la solicitud codificada por la URI. Este flujo (construir un URI RESTful que puede incluir una clave API y recibir una respuesta JSON) es un patrón común para interactuar con servicios externos.

En general, hay (al menos) dos formas de construir llamadas API externas en Ruby: directamente o usando una biblioteca. El método directo sería usar la clase URI en lalibreríaestándar de Ruby para construir el URI de solicitud, usar la clase Net::HTTP para emitir la solicitud a TMDb, verificar si hay errores en la respuesta y, si todo está bien, analizar el objeto JSON resultante. Pero a veces puedes ser más productivos usando una librería (o en el mundo Ruby, una gema).

La gema faraday es una librería HTTP que proporciona una interfaz a través de Net:HTTP. También existe la gema themoviedb-api, que es un "envoltorio" de Ruby aportado por el usuario alrededor de la API RESTful de TMDb, mencionado en las páginas de documentación de la API de TMDb.

Usando la API TMDb directamente

Siempre que te propongas utilizar una API remota, con o sin un contenedor de gemas, debes explorar un poco para aprender cómo funciona la API y, por lo general, también necesitarás obtener una clave de API de desarrollador, incluso antes de usar un contenedor de gema.

- Dedica unos minutos a explorar la versión web interactiva de The Movie DB https://www.themoviedb.org/ para comprender cómo puedes realizar una búsqueda por título de película.

- A continuación, consulta brevemente la documentación de su API https://developer.themoviedb.org/reference/intro/getting-started utilizando las habilidades que aprendistes.
- Solicita una clave API y anótela en un lugar seguro.

Supongamos que tu clave API fuera "5678". Construye un URI que llame a la función de búsqueda de TMDb para buscar la película hecha para televisión "This Life + 10". Aquí hay dos herramientas en línea para ayudarlo a hacer esto https://www.url-encode-decode.com/, https://www.utilities-online.info/urlencode . La función URI.escape de Ruby también puede hacer esto mediante programación, como en URI::escape('https://some.url .com/etc.').

Utiliza el comando curl para emitir esta solicitud a TMDb desde una terminal utilizando tu clave API real.

Usando Faraday

- Antes de comenzar a usar Faraday, asegurate de tener un conocimiento sólido sobre cómo construir URI RESTful para acceder a películas en TMDb. Escribe dos URI que realicen una solicitud GET a TMDb. El primero debe buscar la película Manhunter estrenada en 1986 (es decir, incluir el título de la película así como el año de estreno), y el segundo debe buscar la película Gone Girl con el idioma configurado en inglés.

¿Cómo harías una solicitud GET utilizando uno de los URI definidos en el último paso? ¿Cuál es el tipo de devolución de la solicitud GET de Faraday? ¿Es JSON? Bueno, casi.

Para comprender mejor la estructura del objeto JSON devuelto, realiza algunas llamadas API en tu navegador (simplemente pega el URI en la pestaña de búsqueda) y estudia los resultados. Luego, responde las siguientes preguntas.

- ¿Qué expresión de Ruby devolvería el primer elemento de la lista de películas coincidentes?
- ¿Qué expresión de Ruby establecería la variable overview del primer resultado de búsqueda?
- ¿Qué expresión de Ruby devolvería la fecha de lanzamiento del primer resultado de búsqueda, como un objeto Ruby Date? (Sugerencia: Rails tiene algunas extensiones convenientes para ayudar a administrar objetos de fecha y hora.
- Verdadero o falso: para poder utilizar la API de TMDb desde otro lenguaje como Java, necesitas una librería Java equivalente a la gema ‘themoviedb-api`.
- Verdadero o falso: para poder utilizar la API TMDb de otro lenguaje como Python, necesitas una libreríade Python equivalente a la gema moviedb-api.

Por convención sobre la configuración, las especificaciones para el modelo Movie van en spec/models/movie_spec.rb. Aquí está el camino feliz para llamar a find_in_tmdb en forma de especificación de modelo y la línea de código necesaria en la clase Movie para que se cumpla.

```
require 'rails_helper'

describe Movie do
  describe 'searching Tmdb by keyword' do
    it 'calls Faraday gem with ... domain' do
      expect(Faraday).to receive(:get).with('https://....')
```

```
    Movie.find_in_tmdb('https:….')
   end
  end
end


class Movie < ActiveRecord::Base

  def self.find_in_tmdb(string)
    Faraday.get(string)
  end

  # rest of file elided for brevity
end
```

Ten en cuenta que realmente no te importa si se pasa la URL correcta a Faraday. De hecho, lo que se proporciona ni siquiera fue una URL. El punto es que estás probando si se llaman métodos y no si los métodos devuelven los resultados correctos o arrojan errores.

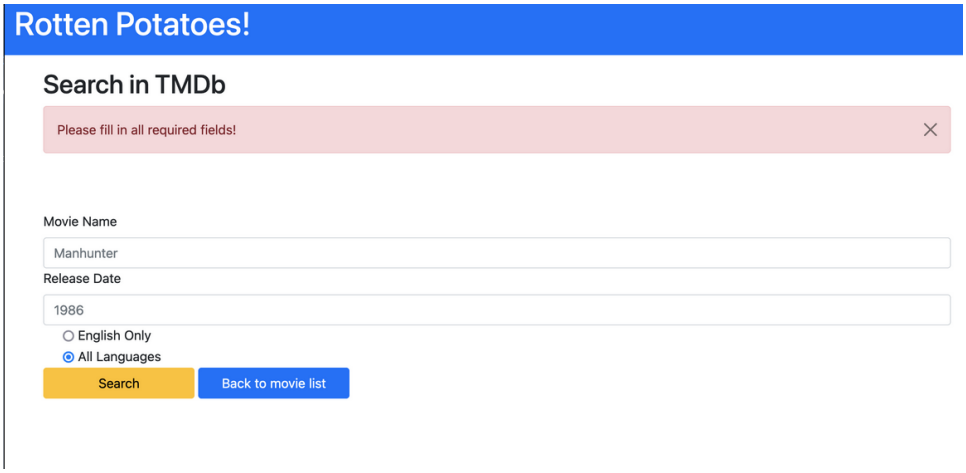
Quizás te preguntes por qué el controlador no simplemente llama al método Faraday.get en sí, en lugar de tener ese método "envuelto" en un método modelo Movie.find_in_tmdb. Hay dos razones. Primero, si la API de la gema de Faraday cambia, tal vez para acomodar un cambio en la API del servicio TMDb, podemos aislar el controlador de esos cambios porque todo el conocimiento de cómo usar la gema para comunicarse con el servicio está encapsulado dentro de la clase Movie. Esta indirección es un ejemplo de cómo separar las cosas que cambian de las que permanecen igual. La segunda y más importante razón es que esta especificación está sutilmente incompleta: find_in_tmdb tiene trabajos adicionales que hacer. Los casos de prueba se han basado en el requisito explícito de que el nombre de una película proporcionado por el usuario debe usarse para consultar TMDb. Pero, de hecho, consultar TMDb requiere una clave API válida.

Paso 5: Paso final (5 puntos)

Observación: El almacenamiento de claves API se incluye en la categoría general de gestión de credenciales y, en versiones anteriores de Rails, se han realizado de diversas formas. Dado que solo tienes una clave API para almacenar, no tomes en cuenta el proceso de proteger tus credenciales en un archivo de entorno, aunque esto es imprescindible para cualquier aplicación real que implementes. Puedes almacenar su clave API como parámetro del método del modelo find_in_tmdb con algún valor predeterminado (posiblemente la clave API correcta). Esto te permitirá sobrescribir la clave API con una no válida cuando estés realizando la prueba.

Si envías tu código a GitHub, asegúrate de que el repositorio esté configurado en modo privado (¡como debería ser!), ya que GitHub detectará automáticamente las claves API expuestas e informará al proveedor. Almacenar claves API simples en un dominio público resultará en la pérdida de la clave API. Si no estás actualizando tu repositorio de GitHub a medida que avanzamos, no deberás preocuparte por esto, sino que lo implementará directamente en Heroku.

Equipado con una clave API, finalmente pasaras a implementar el método find_in_tmdb. Recuerda que solo tienes tres campos de entrada, title de la película, release_year y el language. Ten en cuenta que proporcionar release_year es en realidad opcional, por lo que se puede dejar en blanco y la llamada a la API aún debe realizarse, mientras que si se omite el campo de title, la aplicación debería mostrar un mensaje de error "¡Please fill in all required fields". Puedes hacer este error al verificar el método del controlador.



Además, si no se devuelven películas coincidentes, la aplicación debería mostrar "¡ No movies found with given parameters!".

Como punto de partida, copia el código siguiente en search_tmdb.html.erb. Te asegurará de que los mensajes se muestren en el estilo deseado (bueno, casi, ¿cuál debería ser el valor de la clave?.

```
<% flash.each do |key, value| %>
  <div class="alert alert-<%= key %> alert-dismissible fade show" role="alert">
    <%= value %>
    <button type="button" class="btn-close" data-bs-dismiss="alert" aria-label="Close"></button>
  </div>
<% end %>
```

El método find_in_tmdb debería devolver una lista de películas que no se han guardado en la base de datos. Aquí en realidad no devuelve una calificación MPAA en una consulta de búsqueda de películas. Esto debería hacerse en una llamada API separada, por lo que no se requiere que se muestren las calificaciones correctas y tu debes simplemente poner "R" para todas ellas (para estar seguro).

Stubbing las llamadas API

Desafortunadamente, la especificación que hemos escrito hasta ahora llamará al servicio TMDb real cada vez que se ejecute, lo que hace que la especificación no sea ni Fast (cada llamada tarda unos segundos en completarse) ni Repeteable (la prueba se comportará de manera diferente si TMDb está inactivo o tu computadora no está conectada a Internet). Incluso si solo realizastes pruebas mientras

estaba conectado a Internet, es de muy mala educación que tus pruebas se comuniquen constantemente con un servicio de producción.

Puedes solucionar este problema introduciendo un seam que aísle al que llama desde el destinatario. Una vez más utiliza una librería. WebMock es una libreríade resguardo que usaremos para resguardar las solicitudes al TMDb. Incluya la siguiente línea en el grupo de prueba de Gemfile y ejecuta bundle install.

```
gem "webmock"
```

Luego, necesitas incluir esta librería en spec/spec_helper.rb. Agrega el siguiente código a ese archivo.

```
require 'webmock/rspec'
WebMock.disable_net_connect!(allow_localhost: true)
```

Después de haber hecho esto, asegúrate de incluir spec_helper.rb en las especificaciones de la película llamando a require 'spec_helper'. ¿Qué sucede cuando ejecutamos pruebas de modelos ahora? La única prueba que incluimos aún pasa porque en realidad no estás realizando una llamada a la API, sino que esperas que Faraday haga una llamada, pero no la aplicas, por lo que la llamada nunca se realiza. Agrega una nueva prueba que llame directamente a Movie.find_in_tmdb.

```
it 'calls Tmdb with valid API key' do
  Movie.find_in_tmdb({title: "hacker", language: "en"})
end
```

¿Qué sucede cuando ejecutas la especificación ahora?

En spec_helper.rb continúa y agrega el siguiente bloque.

```
RSpec.configure do |config|
 json_return =
{"page":1,"results":[{"adult":false,"backdrop_path":"/kqMV9VUrGv9BbmRTOzXKyIraeOG.jpg","genre_ids":[],"id":716088,"original_language":"en","original_title":"Sydney 2000 Olympics Opening Ceremony","overview":"Coverage of the glorious Olympic Opening Ceremony of the Games in Sydney. The opening ceremony of the 2000 Summer Olympic games took place on Friday 15 September in Stadium Australia. As mandated by the Olympic Charter, the proceedings combined the formal and ceremonial opening of this international sporting event, including welcoming speeches, hoisting of the flags and the parade of athletes, with an artistic spectacle to showcase the host nation's culture and history.","popularity":3.733,"poster_path":"/nE9GGznpsYPuRIg3kCBgsfCwC2j.jpg","release_date":"2000-09-15","title":"Sydney 2000 Olympics Opening Ceremony","video":false,"vote_average":7.3,"vote_count":3},{"adult":false,"backdrop_path":"/7MMlToGu6JBVtL7myvNQIq0qtIf.jpg","genre_ids":[],"id":716098,"original_language":"en","original_title":"Sydney 2000 Olympics Closing Ceremony","overview":"The 2000 Summer Olympics Closing Ceremony took place on 1 October 2000 in Stadium Australia. The Closing Ceremony attracted 114,714 people, the largest attendance in modern Olympic Games history. IOC President Juan Antonio Samaranch declared
```

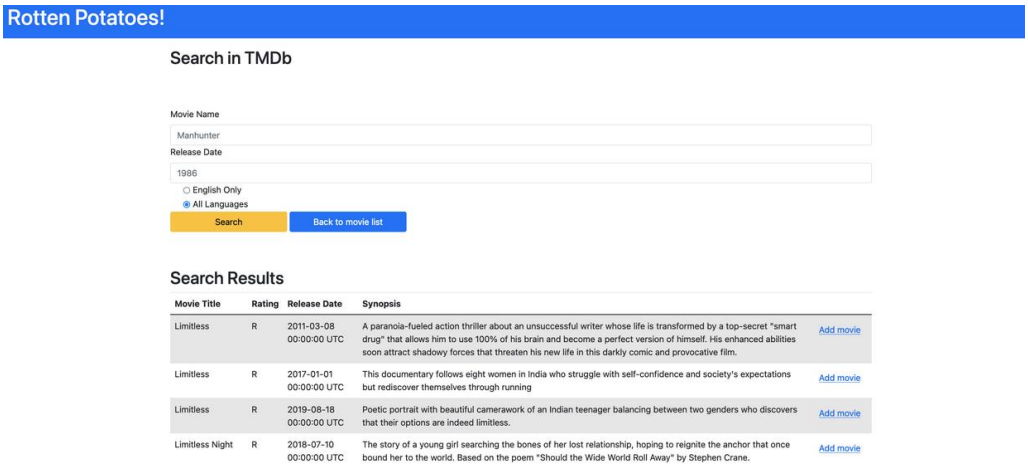
that the 2000 Olympic games were best Olympic Games ever.","popularity":1.308,"poster_path":"/jSxterIFJPP5CDZ4Jderik25hxT.jpg","release_date":"2000-10-01","title":"Sydney 2000 Olympics Closing Ceremony","video":false,"vote_average":10,"vote_count":1}],"total_pages":1,"total_results":2}

```
  config.before(:each) do
    stub_request(:get, /api.themoviedb.org/).
      with(headers: {'Accept'=>'*/*', 'User-Agent'=>'Faraday v1.8.0'}).
      to_return(status: 200, body: JSON.generate(json_return), headers: {})
  end
end
```

Ahora, todas las pruebas deberían estar en verde, porque has bloqueado con éxito la web con una respuesta falsa.

Hay una serie de cosas que aún no se ha probado. Por ejemplo, ¿cómo comprobaría si la clave API es válida o cómo manejaría el caso si no hay películas coincidentes?

Como paso final, se debe mostrar todas las películas devueltas en una tabla. Siguiendo un esquema similar al de la vista index.html.erb, crea una tabla donde irán todos los resultados de la consulta. La tabla que creas debe tener una identificación configurada en search_movies.

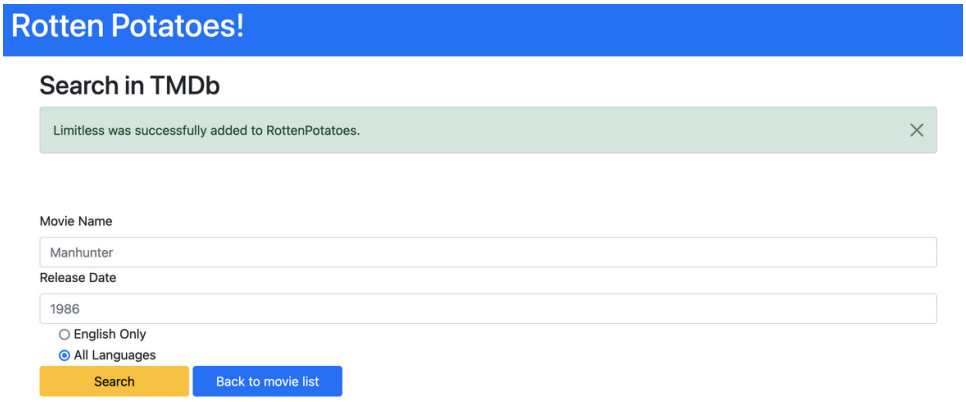


Además, se quiere agregar tener un botón para cada entrada de las películas resultantes para que el usuario pueda agregar estas películas a su base de datos de RottenPotatoes. Dado que este botón modificará la base de datos interna, ¿a qué tipo de solicitud HTTP debería parecerse?

Recuerda que la vista interactúa con el controlador a través de formularios, deberás incluir un formulario para la última columna de la tabla. El formulario debe contener todos los campos necesarios para hacer una película (es decir, título, fecha de lanzamiento, etc.).

Sugerencia: para agregar elementos de entrada sin mostrarlos en la vista, considera usar hidden_field_tag. Dado que este botón Add Movie realiza una solicitud [redactada], deberás crear una ruta y una acción de controlador para ello. Nombra estos add_movie y asegúrate de escribir pruebas como lo hicistes con search_tmdb.

En el método del controlador, guarda el objeto de la película con los parámetros especificados en la base de datos y luego redirije a /search con el mensaje " *movie_name* was successfully added to RottenPotatoes. ".



Vuelve a la página de inicio y asegúrate de que la película agregada aparezca en la lista de películas mostradas. ¿Se seguiría mostrando la película recién agregada si dejas el campo de calificación vacío al crear estos objetos de película por primera vez en Movie.find_in_tmdb?

Y listo, RottenPotatoes ahora puede llamar a la API de TMDb y agregar dinámicamente películas a la base de datos.