Curso de desarrollo de software

Examen Sustitutorio

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. Modificar lo entregado en clase fuera de tiempo en plataformas como GitHub anula tu evaluación.
5. 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 ExamenSustitutorio-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: El ciclo de prueba de aceptación- prueba unitaria

En esta tarea comenzarás con escenarios de Cucumber que fallan, porque estás intentando probar características que aún no se han implementado, y paso a paso, escribirás el código que hace pasar esos pasos. La idea es que estamos agregando una caracteristica de “buscar películas con el mismo director” a RottenPotatoes, y se supone que estos nuevos escenarios probarán esa función.

Corre y prepara rottenpotatoes

Una vez que has utilizado la carpeta dada en la evaluación dado realiza lo siguiente:

1. Cambia al directorio de rottenpotatoes donde vas a desarrollar el ejercicio

2. Ejecuta bundle install --without production para asegurarte de que todas las gemas estén instaladas correctamente.

NOTA: Si Bundler se queja de que está instalada una versión incorrecta de Ruby, verifica que rvm esté instalado (por ejemplo, rvm --version) y escrive rvm list para ver qué versiones de Ruby están disponibles y rvm use para activar una versión particular. Si no hay instaladas versiones que satisfagan la dependencia de Gemfile, puedes decir rvm install version para instalar una nueva versión y luego rvm use para usarla. Luego puedes intentar bundle install nuevamente.

3. Crea el esquema de base de datos inicial
4. Opcionalmente, puedes agregar algunos datos semilla en db/seeds.rb y ejecutar rake db:seed para agregarlos.
5. Verifica que RSpec esté configurado correctamente ejecutando rspec.
6. Verifica que Cucumber esté configurado correctamente ejecutando cucumber. Se proporciona un par de escenarios que fallarán, que puedes usar como punto de partida, en features/movies_by_director.feature.

Lee los mensajes de error de falla de Cucumber. El paso background debería fallar. El primer fallo de prueba en un escenario debería ser: Undefined step: the director of "Alien" should be "Ridley Scott " ¿Qué tendrás que hacer para solucionar ese error específico?

Agrega un campo Director a Movies (1 punto)

Crea y aplica una migración que agregue el campo director a la tabla de películas. El campo director debe ser una cadena que contenga el nombre del director de la película.

Sugerencia: puede resultarle útil el método add_column de ActiveRecord::Migration. Recuerde que una vez aplicada la migración, también debe realizar el bundle exec rake db:test:prepare para cargar el nuevo esquema posterior a la migración en la base de datos de prueba.

- Claramente, ahora que se ha agregado un nuevo campo, tendrás que modificar las Vistas para que el usuario pueda ver e ingresar valores para ese campo. ¿También tienes que modificar el archivo del modelo para que "se note" el nuevo campo? . Muestra con ejemplos tu respuesta
- Mira las definiciones de los pasos fallidos de los escenarios de Cucumber. (¿Dónde encontrará esas definiciones?) Según las definiciones de los pasos, ¿qué pasos del archivo de escenario esperarías aprobar si vuelves a ejecutar Cucumber y por qué? Verifica que los pasos de Cucumber que espera aprobar realmente se aprueben.
- Los pasos background ahora pasan (comprueba esto), pero el primer paso de cada escenario falla, porque le estás pidiendo a Cucumber que "visite una página" pero no has proporcionado una asignación entre el nombre legible por humanos de la página (por ejemplo, the edit page for "Alien") y la ruta real (URL) que iría a esa página en RottenPotatoes. ¿Dónde necesitarás agregar esta asignación (en qué archivo y qué método en ese archivo)?
- Además de modificar las Vistas, ¿tendremos que modificar algo en el controlador? ¿Si es así cuales?
- ¿Qué acciones del controlador, específicamente, no funcionarían si no hicieses el cambio anterior?

Según las autoverificaciones anteriores, deberías poder modificar el controlador y las vistas según sea necesario para estar "consciente" del nuevo campo director.

Utiliza pruebas de aceptación para aprobar nuevos escenarios (2 puntos)

Se proporciona tres escenarios de Cucumber en el archivo features/movies_by_director.feature para impulsar la creación de 2 caminos felices y 1 camino triste de Search for Movies by Director.

El primero te permite agregar información del director a una película existente.

Esta parte no requiere la creación de nuevas vistas o acciones de controlador, pero sí requiere modificar las vistas existentes y requerirá crear una nueva definición de paso y posiblemente agregar una línea o dos a features/support/paths.rb.

Para que el escenario se realice, concéntrate en una línea (paso) a la vez, en orden, y escribe solo el código para que ese paso se realice.

Sugerencia: si deseas concentrarte en ejecutar solo un escenario a la vez, puedes decir cucumber features/movies_by_director.feature:NNN donde NNN es el número de línea inicial (que contiene la palabra clave Scenario:) del escenario específico que deseas ejecutar , o cucumber features/movies_by_director.feature -n "STRING" donde STRING es una subcadena de los nombres de los escenarios que deseas ejecutar (nuevamente según lo especificado en la línea Scenario: de cada uno).

El segundo escenario prueba lo que sucede cuando haces clic en " Find Movies With Same Director " en la página de detalles de una película.

El enlace debería conducir a una nueva página, "the Similar Movies page", que enumera todas las películas que comparten el mismo director que la película mostrada anteriormente.

- Para la primera línea de ese escenario, deberás modificar la vista existente Show Movies (detalles) para agregar el enlace. Ten en cuenta que puede realizar este paso siempre que el destino del enlace sea una URL válida. No es necesario que configures la página objetivo real todavía. Esto es parte del proceso BDD, lo que te permite realizar un paso a la vez: puedes concentrarte en asegurarte de que el enlace aparezca y se pueda hacer clic utilizando cualquier URL válida, luego continúe con la creación de la página correcta para que llegue (según sea necesario para que se apruebe el siguiente paso del escenario).
- Deberás elegir un nombre de ruta RESTful para esta nueva acción (“show movies with similar director”) y deberás agregar una ruta, vista y método de controlador para ello. Una ruta sugerida podría ser GET /movies/:id/similar donde :id es un parámetro de ruta que coincidirá con el ID de la película cuyo director deseas que coincida. (Sugerencia: dado que esta ruta no se crea como una de las rutas predeterminadas por rersources: movies, verifica la sintaxis de " Non-Resource-Based Routes " en la documentación de Rails Routing- https://guides.rubyonrails.org/routing.html).
- Necesitarás un método en el modelo Movie para encontrar películas cuyo director coincida con el de la película actual. La acción del controlador que maneja la nueva ruta llamará a ese método modelo.

¿Este método modelo sería un método de clase o un método de instancia?

- Para que el escenario Cucumber encuentre la nueva página, probablemente también necesitarás modificar features/support/paths.rb para reconocer el nombre en “lenguaje sencillo” de Cucumber de esta nueva ruta, como se indica en el paso del escenario " I should be on the Similar Movies page for……".

El tercer escenario aborda el camino triste.

Cuando la película actual no tiene información del director pero intentamos hacer " Find with same director " de todos modos, debemos redirigir a la página de la lista de todas las películas y mostrar un mensaje útil.

Sugerencia: Utiliza flash[] para almacenar un mensaje y, en la lista de todas las películas, muestra condicionalmente el valor de flash[] apropiado si no está vacía. El uso de las clases Bootstrap CSS alert alert-info o alert alert-warning en el elemento de visualización de mensajes lo mostrará en un cuadro resaltado en color.

El éxito de esta parte es lograr que se cumplan todos los escenarios.

Cobertura del código (1 punto)

Hay muchas maneras de medir la cobertura del código, pero este ejercicio ha preinstalado una librería llamada SimpleCov que mide la "cobertura de declaraciones", esencialmente, qué líneas de tu aplicación fueron al menos tocadas por sus pruebas.

Puedes ver que las dos primeras líneas de features/support/env.rb (que Cucumber carga automáticamente en cada ejecución) inician la medición de cobertura de prueba de SimpleCov.

Cada vez que ejecuta tus pruebas (ya sean pruebas de aceptación de Cucumber o pruebas de unidad/módulo RSpec), SimpleCov genera un informe en un directorio llamado coverage/. Para ver los resultados, abre coverage/index.html después de una ejecución de prueba y haz clic en el nombre de cualquier archivo en tu aplicación para ver qué líneas cubrieron tus pruebas.

¿Qué escenarios de Cucumver tendrías que agregar para cubrir las líneas no cubiertas? Si la cobertura de su prueba aún no es del 90%, agrega escenarios y pasos para cubrir los huecos como indica SimpleCov. Obtendrás el 100 % del puntaje de este paso si obtienes al menos un 95 % de cobertura de código.

Parte 2: Ruby on Rails

Pregunta 1 (1 punto)

¿Por qué la abstracción de un objeto de formulario pertenece a la capa de presentación y no a la capa de servicios (o inferior)?

Pregunta 2 (1 punto)

¿Cuál es la diferencia entre autenticación y autorización?

Pregunta 3 (2 puntos)

Un middleware es un componente que envuelve la ejecución de una unidad central (función) y puede inspeccionar y modificar datos de entrada y salida sin cambiar su interfaz. El middleware suele estar encadenado, por lo que cada uno invoca al siguiente y sólo el último de la cadena ejecuta la lógica central. El encadenamiento tiene como objetivo mantener el middleware pequeño y de un solo propósito.

Un caso de uso típico del middleware es agregar registro, instrumentación o autenticación. El patrón es popular en la comunidad Ruby y, aparte de Rack, lo utilizan Sidekiq, Faraday, AnyCable, etc. En el mundo que no es Ruby, el ejemplo más popular sería Express.js.

Rails incluye más de 20 **middlewares** por defecto. Puedes ver la pila de middleware ejecutando el comando bin/rails middleware: bin/rails middleware

Aprendimos que manejar una solicitud web implica miles de llamadas a métodos y objetos Ruby asignados. ¿Qué pasa si omite el middleware de Rack y se pasa la solicitud al enrutador directamente (Rails.application.routes.call(request))? ¿Qué pasa si se omitie el enrutador y llamar a una acción del controlador de inmediato (por ejemplo, PostsController.action(:index).call(request))?

La gema trace_location (https://github.com/yhirano55/trace_location) es el pequeño ayudante de un desarrollador curioso. Su objetivo principal es aprender qué sucede detrás de escena de las API simples proporcionadas por librerías y frameworks. Te sorprenderá lo complejos que pueden ser los aspectos internos de las cosas que das por sentado (por ejemplo, user.save en Active Record).

Diseñar API simples que resuelvan problemas complejos es un verdadero dominio del desarrollo de software. En el fondo, esta joya utiliza la API TracePoint de Ruby (https://rubyapi.org/3.2/o/tracepoint), una poderosa herramienta de introspección en tiempo de ejecución.

Utiliza trace_location para realizar algunos experimentos y analizar los resultados.

Pregunta 4 (2 punto)

Active Record es la parte más grande de Rails, su código base contiene el doble de archivos (más de 1000) y líneas de código (más de 100 000) que el segundo más grande, que es Action Pack. Con esa cantidad de implementaciín, proporciona docenas de API para que los desarrolladores las utilicen en sus aplicaciones. Como resultado, los modelos heredados de Active Record tienden a conllevar muchas responsabilidades. Estas clases Ruby excesivamente responsables generalmente se denominan Gods objects.

Desde la perspectiva del código, mucha responsabilidad significa muchas líneas de código fuente. El número de líneas en sí no puede considerarse un indicador de código en mal estado. Necesitamos mejores métricas para identificar buenos candidatos para refactorizar el código base.

Se ha demostrado que la combinación del churn y la complejidad son tales indicadores.

El churn describe la frecuencia con la que se ha modificado un archivo determinado. Una tasa de cambio alta podría indicar una falla en el diseño del código: o estamos agregando nuevas responsabilidades o estamos tratando de eliminar las deficiencias de la implementación inicial.

Hoy en día, cada proyecto utiliza un sistema de control de versiones, por lo tanto, podemos calcular el churn de un archivo determinado como el número total de commits en las que el archivo se ha visto afectado. Con Git, el siguiente comando devuelve el factor churn para el modelo de User:

git log --format=oneline -- app/models/user.rb | wc -l

Podemos ir más allá y usar el poder de los comandos de Unix para obtener los 10 archivos según el churn:

find app/models -name "*.rb" | while read file; do echo $file 'git log --format=oneline -- $file | wc -l'; done | sort -k 2 -nr | head

Dependiendo de la antigüedad del proyecto, es posible que desees limitar el rango de fechas de commits(utilizando la opción --since).

Con respecto a la complejidad, no existe un algoritmo único de complejidad del código para calcular las métricas correspondientes. En Ruby, una herramienta llamada Flog (https://github.com/seattlerb/flog) es el instrumento de facto para calcular la complejidad del código fuente.

Así es como podemos obtener la complejidad general de un solo archivo:

flog -s app/models/user.rb

Armados con estas dos métricas ahora podemos definir la regla general para identificar las clases de Ruby que merecen la mayor atención de refactorización.

Prepara un one-liner (https://linuxcommandlibrary.com/basic/oneliners) de Unix para mostrar los N primeros archivos complejos de Ruby usando Flog. ¿Puedes combinarlo con la calculadora de churn para mostrar los N archivos por churn* complejidad?.

Sugerencia: attractor (https://github.com/julianrubisch/attractor) es una herramienta de visualización y cálculo de la complejidad del código. Calcula tanto el churn como la complejidad (usando Flog) y proporciona una interfaz web interactiva para analizar los datos recopilados. Soporta Ruby y JavaScript, siendo así una solución completa para aplicaciones web Rails.

Parte 3: JavaScript

Pregunta1 (2 puntos)

Crea varias funciones que te permitirán interactuar con las cookies de la página, incluida la lectura de un valor de cookie por nombre, la creación de una nueva cookie usando un nombre y su configuración para una cantidad determinada de días, y la eliminación de una cookie.

Configura tu página web y, en el código JavaScript, genera el valor de documento.cookie que debería estar en blanco. Intenta eliminar un cookie por su nombre.

Pregunta 2 (1 punto)

Este ejercicio es un ejemplo de una estructura de formulario típica en la que se verifican los valores ingresados en el formulario y se validan antes de enviar el contenido. Se devuelve una respuesta al

usuario si los valores no cumplen con los criterios de validación en el código. Utiliza el siguiente HTML y CSS como plantilla inicial:

```
<!doctype html>
<html>
<head>
	<title>Curso CC-3S2</title>
	<style>
		.hide {
			display: none;
		}
		.error {
			color: red;
			font-size: 0.8em;
			font-family: sans-serif;
			font-style: italic;
		}
		input {
			border-color: #ddd;
			width: 400px;
			display: block;
			font-size: 1.5em;
		}
	</style>
</head>
<body>
	<form name="myform"> Email :
		<input type="text" name="email"> <span class="error hide"></
span>
	<br> Password :
```

```
        <input type="password" name="password"> <span class="error
hide"></span>
                <br> User Nombre :
                <input type="text" name="userName"> <span class="error hide"></
        span>
                <br>
                <input type="submit" value="Sign Up"> </form>
        <script>
        </script>
</body>
</html>
```

Tome los siguientes pasos:

1. Usando JavaScript, selecciona todos los elementos de la página y configúralos como objetos JavaScript para que sean más fáciles de seleccionar dentro del código. Selecciona también todos los elementos de la página que tengan la clase **error** como objeto.
2. Agrega un detector de eventos para enviar y capturar el clic, evitando la acción del formulario predeterminado.
3. Recorre todos los elementos de la página que tienen la clase **error** y agrega la clase **ocultar,** lo que los eliminará de la vista ya que se trata de un envío nuevo.
4. Utilizando la expresión regular para correos electrónicos válidos, prueba los resultados con el valor de entrada del campo de correo electrónico.
5. Crea una función para responder a errores, que elimine la clase **oculta** del elemento junto al elemento que desencadenó el evento.
6. Si hay un error de que una entrada no coincide con la expresión regular deseada, pasa los parámetros a la función de manejo de errores que acabas de crear.
7. Verifica el valor de entrada del campo de contraseña para asegurarye de que solo se utilicen letras y números. También verifica la longitud para asegurarse de que tenga entre 3 y 8 caracteres. Si alguno de ellos es falso, agrega el error con la función error y crea un mensaje para el usuario. Establece el error booleano en verdadero.
8. Agrega un objeto para realizar un seguimiento de la creación de datos del formulario y agrega valores al objeto recorriendo todas las entradas, configurando el nombre de la propiedad para que sea el mismo que el nombre de la entrada y el valor igual que el valor de la entrada.
9. Antes de finalizar la función de validación, verifica si todavía hay un error y, si no es así, envía el objeto de formulario.

Pregunta 3 (2 punto)

Extienda la función de validación en ActiveModel que se ha utilizado en actividades de clase para generar automáticamente código JavaScript que valide las entradas del formulario antes de que sea enviado. Por ejemplo, puesto que el modelo Movie de RottenPotatoes requiere que el título de cada película sea distinto de la cadena vacía, el código JavaScript debería evitar que el formulario "Add New Movie" se enviara si no se cumplen los criterios de validación, mostrar un mensaje de ayuda al usuario, y resaltar el(los) campo(s) del formulario que ocasionaron los problemas de validación. Gestiona, al menos, las validaciones integradas, como que los títulos sean distintos de cadena vacía, que las longitudes máxima y mínima de la cadena de caracteres sean correctas, que los valores numéricos estén dentro de los límites de los rangos, y para puntos adicionales, realice las validaciones basándose en expresiones regulares.

Parte 4 (2 puntos)

Para este ejercicio utiliza la actividad que has desarrollado en clase. supongamos que está probando un nuevo método de instancia de la clase Movie llamado name_with_rating que devuelve una cadena con un formato agradable que muestra el título y la calificación de una película. Claramente, dicho método tendría que acceder a title y a los atributos rating de una instancia Movie. Crea un doble que conozca toda esa información y pasar ese doble:

```
fake_movie = double('Movie')
allow(fake_movie).to receive(:title).and_return('Casablanca')
allow(fake_movie).to receive(:rating).and_return('PG')
expect(fake_movie.name_with_rating).to eq 'Casablanca (PG)'
```

Pero dado que el método de instancia que se está probando es parte de la clase Movie en sí, tiene sentido usar un objeto real aquí, ya que no se trata de aislar el código de prueba de las clases colaboradoras.

Pregunta: ¿Dónde podemos conseguir una instancia de Movie real para utilizarla en dicha prueba?

Pregunta: ¿Qué hacen los siguientes códigos entregados y donde se ubican? . ¿Por qué hay que tener cuidado en su uso?. Comprueba tus respuestas.

```yaml
# spec/fixtures/movies.yml
milk_movie:
    id: 1
    title: Milk
    rating: R
    release_date: 2008-11-26

documentary_movie:
    id: 2
    title: Food, Inc.
    release_date: 2008-09-07
```

```ruby
# spec/models/movie_spec.rb:

require 'rails_helper.rb'

describe Movie do
    fixtures :movies
    it 'includes rating and year in full name' do
        movie = movies(:milk_movie)
        expect(movie.name_with_rating).to eq('Milk (R)')
    end
end
```

Al terminar analiza los pros y los contras del uso de factories o fixtures en las pruebas.

Parte 4: Pruebas y Rspec (3 puntos)

El sistema de puntuación utilizado en el tenis sobre hierba se originó en la Edad Media. A medida que los jugadores ganan puntos sucesivos, sus puntuaciones se muestran como 15, 30 y 40. El siguiente punto es una victoria a menos que tu oponente también tenga 40. Si ambos están empatados a 40, entonces se aplican reglas diferentes: el primer jugador con una clara ventaja de dos puntos es el ganador. Algunos dicen que el sistema 0, 15, 30, 40 es una corrupción del hecho de que la puntuación solía hacerse utilizando los cuartos de un reloj.

Se quiere escribir una clase que maneje este sistema de puntuación. Utiliza las especificaciones RSpec para realizar el proceso. Utiliza el siguiente archivo de especificación:

```
RSpec.describe "TennisScorer" do

  describe "puntuación básica" do

    it “empieza con un marcador de 0-0"

    it “ hace que el marcador sea 15-0 si el sacador gana un punto"

    it "hace que el marcador sea 0-15 si el receptor gana un punto"

    it "hace que el marcador sea 15-15 después de que ambos ganen un punto"

  end

end
```

1. Ejecuta esta especificación usando el comando rspec.
2. Escribe un archivo .rb para cumplir la primera expectativa. Recuersa que como muchos Ruby DSL, RSpec aprovecha la flexibilidad de Ruby para generar un código que no se parece exactamente al Ruby normal. Al intentar comprender RSpec, resulta útil restablecer los

paréntesis completos y los receptores de mensajes self implícitos como guía de lo que realmente está sucediendo.

3. También puedes ver que la expectativa real también es solo un conjunto de llamadas a métodos. El método expect se llama con un objeto como argumento. El resultado de ese método se pasa al método to, que a su vez toma un argumento que se genera llamando a eq. El resultado de la llamada a eq es un matcher, y RSpec define una serie de matchers que interactúan con el método to (o su método hermano not_to) para determinar si la expectativa se cumple o no. Configura la clase TennisScorer, pero sólo lo suficiente para que satisfaga esta aseveración. Ahora puedes ejecutar la especificación nuevamente.
4. Ahora sólo tenemos tres especificaciones pendientes. Escribe las siguientes especificaciones (agrega una nueva para un caso de error). Esto se aprueba ?.Rectifica este caso.
5. El código no está terminado, pero ahora se aprobarán las especificaciones existentes. Ahora ejecuta la especificación nuevamente. Debes estar cumpliendo dos de las cuatro expectativas iniciales. Corrige la duplicación en la especificación, al mismo tiempo, desarrolla las dos últimas expectativas.
6. RSpec nos brinda una forma alternativa y preferida de configurar variables que son condiciones para las pruebas. El método let crea lo que parece una variable cuyo valor se obtiene al evaluar un bloque. Escribe el script correspondiente
7. Escriba alguna de las siguientes expectativas y continúa desarrollándo el ejercicio.

It “40-0 después de que el sacador gane tres puntos"

It “W-L después de que el sacador gana cuatro puntos"

It “L-W después de que el receptor gane cuatro puntos"

It “Deuce después de cada uno gana tres puntos"

It “El sacador con ventaja después de cada uno gana tres puntos y el sacador obtiene uno más".