Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ciencia de la Computación

IIC3745 — Testing
2024 - 1

Tarea 3

Fecha de entrega: Martes 4 de Junio del 2024 a las 23:59

Información General

La siguiente tarea contempla como objetivo principal el aplicar dos de los niveles de testing descritas en el modelo V a una aplicación web: unit testing e integration testing. Por lo tanto, el grupo podrá aplicar de manera práctica los conceptos vistos en cátedra.

Objetivos

- Profundizar conocimientos del framework Ruby on Rails para desarrollar aplicaciones web aplicando técnicas de testing.
- Profundizar conocimientos sobre herramientas y buenas prácticas de desarrollo de software.
- Aplicar conocimientos adquiridos durante el curso para asegurar calidad de un software.

Contexto

En los últimos años, ha habido un creciente interés en artículos deportivos y el alquiler de canchas. Como respuesta a esta demanda, DCCorporation se propuso crear DCCancha, una aplicación web que facilita el alquiler de canchas y la compra de artículos deportivos. Después de varios meses de desarrollo, DCCorporation finalizó la primera versión del proyecto. Sin embargo, el equipo de desarrollo no siguio buenas prácticas de pruebas, ya que no implementaron ningún test para asegurar la calidad y estabilidad de la aplicación.

DCCorporation preocupada de que puedan existir defectos en la aplicación que generen una enorme perdida para el negocio, se contacta con ustedes para que desarrollen un plan de pruebas que permita entregar tranquilidad sobre la calidad y estabilidad de la aplicación.

Estado actual de la aplicación

A continuación se detallan aspectos sobre el estado actual de DCCancha para que tengan en cuenta al momento de realizar las pruebas.

- *Landing page.* La aplicación cuenta con un landing page donde cualquier visitante podrá revisar la información que ofrece la página.
- *Listar productos.* En el navbar se cuenta con una sección de productos. Por lo tanto, un usuario tiene acceso a una lista de todos los productos que se ofrecen, además de los detalles específicos de cada producto. También, es posible filtrar los productos por nombre y categoría. Para productos que sean canchas el sitio permite que usuarios registrados realicen solicitudes de reserva. Para el resto de los productos se entrega la opción de agregar los productos al carro de compras.
- *Escribir mensajes y reseñas.* Los usuarios registrados pueden dejar preguntas para el vendedor y también reseñas de los productos.
- *Administrar productos, mensajes y reseñas.* Los administradores pueden crear, editar y eliminar productos de la pagina. También pueden responder las preguntas de los productos y moderar tanto las preguntas como las reseñas pudiendo modificarlas o eliminarlas.
- *Contactar a la administración.* El navbar cuenta con una sección de contacto donde cualquiera de los usuarios puede dejar un mensaje a la administración del sitio.
- *Revisar carrito de compra.* Un usuario registrado puede agregar productos a su carro de compras. Además, el navbar permite que a un usuario revisar el detalle de lo que se encuentra en el carro junto con la opción de pagar.
- *Pagar carrito de compra.* El sitio simula al pagar todos los pasos de una compra online real pero saltándose la parte del pago con el banco.
- *Ver lista de deseos.* Un usuario registrado cuenta con una lista de deseos en la cual podrá agregar o eliminar productos que le interesen.
- *Administrar perfil y solicitudes.* Un usuario registrado puede actualizar la información de su perfil, ver las solicitudes de compra y reserva que ha realizado o que se encuentran en curso. En caso de que se encuentre en curso, se da la posibilidad de aceptar o rechazar las reservas solicitadas a las canchas que ese usuario publico.

En el README del proyecto pueden encontrar más detalles. Además, se describe algunas decisiones de diseño del equipo de desarrollo.
La empresa les da completa libertad para que modifiquen lo que quieran de la página pero esta deberá seguir cumpliendo con las funcionalidades y requerimientos originales. Además, en caso de agregar otra funcionalidad, se espera que esta funcione según las especificaciones de un requerimiento.

Diagrama entidad relación

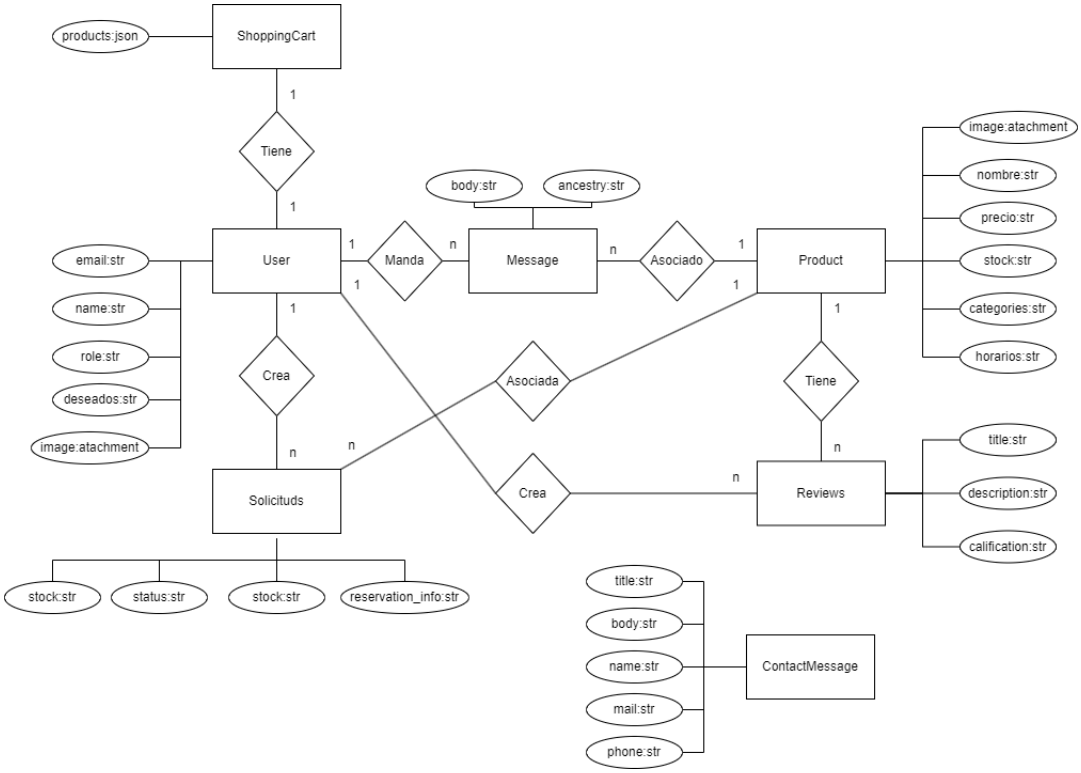

Figura 1: Diagrama de Entidad-Relación inicial

Tareas a realizar

Recomendamos fuertemente que realicen los siguientes pasos en orden:

- Realizar el setup del proyecto y subirlo a un repositorio privado suyo. **Importante:** Tengan cuidado de copiar todos los archivos, ya que hay algunos que son ocultos y no aparecen por default en su navegador de archivos (por ejemplo .rubocop.yml).
- Configurar su repositorio y definir la metodología de desarrollo que van a ocupar.
- Crear la integración continua para automatizar rubocop y la ejecución de los tests. Se deberá almacenar como artefacto el coverage que se obtenga después de ejecutar los tests. En la sección integración, se detalla las características que deberá cumplir su integración continua.
- En paralelo crear tests unitarios sobre los modelos y tests de integración sobre los controladores. Podran usar MiniTest que es la gema por default o la gema rspec para crear sus tests. También es posible agregar la gema FactoryBot para simplificar la creación de instancias en sus tests. En las próximas secciones (pruebas unitarias y pruebas de integración) se detallan en especifico en que deben focalizarse y se entregan consejos que les pueden ser útiles.
- Obtener un 100% de coverage en los modelos y controladores pedidos. El proyecto ya se encuentra configurado para calcular el coverage utilizando simplecov después de cada ejecución de los tests. En caso de identificar alguna sección del código que sea inalcanzable podrán argumentar en el readme las razones y realizar modificaciones en el código ya sea eliminado o modificando los pedazos de código para que sean alcanzables según sea el caso. En caso de entregar una buena argumentación no se descontara por esa sección no cubierta. **Importante:** Tengan cuidado al modificar, ya que pueden introducir nuevos defectos a la aplicación o eliminar features de esta.

Integracion Continua

Deberán crear **un solo workflow file** que contenga **dos jobs**. Uno de los jobs deberá ejecutar rubocop y el otro deberá ejecutar los tests. Al finalizar la ejecución es necesario almacenar toda la carpeta coverage como artefacto. Ambos jobs se deberán iniciar al realizar pull request a la rama main. En caso de utilizar una herramienta distinta a github actions deberán cumplir con la misma idea dado los términos de la herramienta que elijan.
Para quienes tengan experiencia con github actions, se recomienda que aprovechen esta oportunidad de aprender sobre otras herramientas, como por ejemplo CircleCI.

Vocabulario

Antes de hablar de los tests, definiremos dos conceptos a los cuales haremos referencia en esta tarea:

- *Test/Caso happy path*: Caso de prueba en el que se espera que se siga el flujo ideal, en otras palabras lo que hacemos resulta exitoso. Por ejemplo, se instancia un modelo valido o se manda un request valido. En estos casos generalmente lo que se espera es que la acción se realice correctamente y la pagina reaccione sin lanzar alertas o errores. Por lo tanto los tests fallan cuando lo anterior no se cumple.
- *Test/Caso de camino alternativo*: Caso de prueba en el que después de ejecutar la acción se espera que se levante alguna alerta o error. Por ejemplo, tratamos de instanciar un modelo invalido, cometemos un error en el request o nos falta autorización. En estos casos generalmente lo que se espera es que la acción que se quería realizar no se realice y que la pagine genere algún tipo de mensaje o alerta. Por lo tanto, los tests fallan solo si la acción se realiza exitosamente y/o no se genera ninguna alerta o error.

Tests unitarios

Deberán realizar tests unitarios sobre todos los modelos de la aplicación ubicados en app/models.
El objetivo es revisar que los modelos que se instancien solo se hagan con información valida. Por lo tanto deberán testear que cada modelo cuente con las validaciones necesarias. Además, si el modelo tiene algún método implementado (por ejemplo calcular el total del carro) también deberán asegurarse de que su funcionamiento sea correcto.

Condiciones: Como mínimo se espera que para cada modelo se pruebe un caso happy path y para cada validación que tenga el modelo realicen uno o más tests de camino alternativo dependiendo del tipo de validación. Deberán crear instancias de los modelos y verificar si estas son validas o no según los datos que se ingresen.

Consejos:

- Tengan cuidado con la diferencia entre el método .create y el método .new o .build ya que el primero crea y guarda la instancia en la base de datos pero el segundo solo la crea sin guardarla.
- Es importante que con instancias que no sean validas no ocupen .create ya que como no van a pasar las validaciones les van a arrojar errores. Para verificar si una instancia es valida o no, pueden usar el método .valid? en sus asserts.
- Tengan cuidado de no guiarse por el coverage al construir los tests sobre los modelos, ya que en rails es fácil obtener el 100% solo con el happy path. Si no incluimos los tests de camino alternativo podríamos estar permitiendo combinaciones de datos inválidos.
- Recuerden el principio "el testing exhaustivo no es posible". Por lo tanto si uno de los campos es un string es imposible probar todas las combinaciones de inputs, sin embargo, es posible identificar algunos casos relevantes o que son equivalentes a otros para armar los tests.
- Ojo con el modelo user, que importa algunas validaciones de devise, como ser la existencia del correo y contraseña, o que la contraseña sea la misma que la confirmación entre otras cosas.
- Para depurar sus tests unitarios pueden revisar la lista de errores después de validar su instancia con el método .errors.full_messages el cual lo pueden imprimir al ejecutar.

Tests de integracion

Deberán realizar tests de integración sobre todos los controladores de la aplicación revisando el correcto funcionamiento de cada endpoint.
Nos centraremos en revisar los mensajes de alerta que la aplicación entregue almacenados en la variable *notice* y los cambios que ocurran en la base de datos.

Condiciones: Como mínimo se espera que prueben lo siguiente dependiendo del tipo de endpoint:

- *Endpoint GET:* Un caso happy path donde revisen que el estado que se obtenga es un ok (código 200) y en caso de que exista alguna protección de autentificaron casos alternativos para cubrirlos.
- *Endpoint POST:* Un caso happy path revisando que la base de datos se modifique correctamente y que incluya el mensaje correcto en el notice en caso de tenerlo implementado. También incluir al menos un test de camino alternativo para probar casos en que la operación no sea exitosa revisando que la operación no se concrete en la base de datos y además se incluya el mensaje correcto en el notice. No es necesario que pruebe todas las combinaciones de datos inválidos ya que consideraremos que eso fue cubierto con los tests unitarios.

- *Endpoint PATCH:* Lo mismo que para los endpoints POST. Recuerde que si en sus tests guarda en una variable alguna instancia de algún modelo y la modifica en algún request tiene que ir a buscar esa instancia nuevamente en la base de datos o utilizar .reload para actualizar los datos de la instancia antes de revisar los atributos en sus asserts.
- *Endpoint DELETE:* Un caso happy path en donde la operación sea exitosa revisando que efectivamente se elimine la instancia de la base de datos y se entregue el mensaje correspondiente en el notice. Además incluir los caminos alternativos necesarios para revisar los casos en que la operación no sea exitosa asegurándose que no se realicen cambios en la base de datos y que se entregue el mensaje correspondiente.

Consejos:

- Solamente es necesario revisar el response code de los endpoint GET ya que la aplicación generalmente redirecciona en todos los otros casos. Los permisos que tienen los usuarios están definidos en `app/models/ability.rb` usando la gema CanCan.
- Para depurar sus tests después de realizar un request se guarda en la variable response toda la información que se recibió del request. También podrán revisar la información que esta contenida en la variable notice para entregarles una idea del problema.

Resumen: Objetos a testear

En caso de encontrar defectos repórtenlos en el readme y los pueden arreglar si quieren.

Modelos	Controladores
contact_message	contact_message_controller
message	message_controller
product	products_controller
review	review_controller
shopping_cart	shopping_cart_controller
solicitud	solicitud_controller
user	user_controller
	pages_controller

Entrega

La entrega de esta tarea se realizara en github. Para la entrega, en su repositorio **privado** deberán crear la rama T3 desde main la cual no deberá ser modificada posterior a la fecha de entrega de tarea. Se controlara el ultimo commit de la rama T3 para identificar entregas atrasadas. Además de lo anterior, deberán enviar en el buzón de la tarea la url a su repositorio.
Nota: Una semana antes de la fecha de entrega se publicara un anuncio con el detalle de a quienes deberán invitar a su repositorio para la corrección.

Criterio de Corrección

A continuación, se presenta la distribución de puntaje que se usará para evaluar esta tarea.

- Integración Continua [1 pts].
- Tests unitarios sobre los modelos [2 pts].
- Test de integración sobre los controladores [2 pts].
- Porcentaje de coverage [1 pts].

Descuentos

Los revisores podrán aplicar descuentos a las notas de sus entregas:

- Hasta 5 décimas a criterio del revisor si realizan algo que dificulte la corrección de su entrega y obligue al revisor a demorarse más tiempo de lo necesario en corregir. Por ejemplo: No especificar algo importante en el `README` sobre su código, modificar las variables de entorno para conectarse a la base de datos o casos similares.
- Descuento de hasta un punto si se descubre que alguna de las funcionalidades originales de la aplicación ya no funciona en la entrega.

Consideraciones Generales

- Ustedes deberán crear su propio repositorio en GitHub **privado** y configurarlo a su gusto para cumplir con lo que se pide en la entrega. También, deberán darle acceso a su repositorio a algunos usuarios. La información sobre los usuarios se les informara a través de un anuncio en canvas una semana antes de la entrega.
- Para cada entrega deberán actualizar el `README` que se encuentra dentro del proyecto. En este archivo tendrán que señalar lo que lograron hacer, quien hizo cada cosa e incluir cualquier información de su aplicación que pueda ser útil o necesaria para el corrector por ejemplo de que forma invocar los tests.

Reportar problemas en el equipo

En el caso de que algún integrante no aportara como fue esperado en la tarea, podrán reportarlo enviando un correo con asunto **Problema Equipo {NumeroGrupo} Testing** a *nicolas.olmos@uc.cl* explicando en detalle lo ocurrido. Posterior a eso se revisara en detalle la contribución del equipo y se analizara si corresponde aplicar algún descuento. Instamos a todas las parejas que mantengan una buena comunicación y sean responsables con el resto de su equipo para evitar problemas de este estilo.
Advertencia: Si algún integrante del grupo no aporta en las tareas recibirá la nota mínima en esa entrega.

Restricciones y alcances

- En caso de dudas con respecto al enunciado o código base, deberán realizarlas en un foro relacionado a la tarea que se encontrara disponible en canvas.
- El uso de gemas adicionales deberá ser consultado previamente en el foro a excepción de la gema FactoryBot que si puede ser incluida.

Atraso

Cada grupo cuenta con un cupón de atraso. Si el grupo decide activarlo en una tarea podrán realizar la entrega dos días después de la fecha original de entrega. Para ocuparlo solo deben realizar una entrega pasado la fecha señalada en el enunciado. Realizar la entrega atrasada sin contar con un cupón para usar implicara obtener nota mínima en la entrega.

Integridad académica

Este curso se adscribe al Código de Honor establecido por la Escuela de Ingeniería. Todo trabajo evaluado en este curso debe ser hecho **individualmente** o en **los grupos asignados** según sea definido en la evaluación y **sin apoyo de terceros**. Se espera que los alumnos mantengan altos estándares de honestidad

académica, acorde al Código de Honor de la Universidad. Cualquier acto deshonesto o fraude académico está prohibido; los alumnos que incurran en este tipo de acciones se exponen a un Procedimiento Sumario. Es responsabilidad de cada alumno conocer y respetar el documento sobre Integridad Académica publicado por la Dirección de Pregrado de la Escuela de Ingeniería.

¡Éxito! :)