**UNIVERSIDAD NACIONAL AGRARIA DE LA SELVA**

**FACULTAD DE INGENIERÍA EN INFORMÁTICA Y SISTEMAS**

**Escuela Académica Profesional De Ingeniería En Informática Y Sistemas**

Diagrama

Descripción generada automáticamenteLogotipo

Descripción generada automáticamente

**DOCUMENTACIÓN DE LAS PRUEBAS REALIZADAS AL SOFTWAR ShopMen´s**

**AUTORES : Tarazona Narciso Nyels Renan**

**Aguirre Valdez Denilson**

**ASIGNATURA : Pruebas de Software**

**DOCENTE : Pando Bryan Cesar**

**Tingo María – Perú**

**Marzo 2024**

**ÍNDICE**

[1 INTRODUCCIÓN 3](#_Toc173230283)

[2 PRUEBAS DE CAJA BLANCA 3](#_Toc173230284)

[2.1 COBERTURA DE SENTENCIAS 3](#_Toc173230285)

[2.2 COBERTURA DE RAMAS 3](#_Toc173230286)

[2.3 COBERTURA DE DESICIONES 4](#_Toc173230287)

[2.4 COBERTURA DE CAMINOS 5](#_Toc173230288)

[2.5 CAMINO MÁS CORTO 6](#_Toc173230289)

[3 PRUEBAS DE CAJA NEGRA 6](#_Toc173230290)

[3.1 PARTICIÓN DE EQUIVALENCIAS 6](#_Toc173230291)

[3.2 PRUEBAS DE VALOR LÍMITE 7](#_Toc173230292)

[3.3 TABLAS DE DECISIÓN 8](#_Toc173230293)

[3.4 CASOS DE USO 8](#_Toc173230294)

[3.5 TRANSICIÓN DE ESTADO 9](#_Toc173230295)

# INTRODUCCIÓN

En el presente informe se detallan las actividades y resultados de las pruebas realizadas al software de una tienda de ropa en línea destinada al público masculino. Este proyecto tiene como objetivo desarrollar una plataforma robusta y eficiente que permita a los usuarios buscar, seleccionar y comprar productos de manera fácil y segura.

El objetivo principal de este informe es documentar el proceso de pruebas realizado para asegurar que todas las funcionalidades del software cumplan con los requisitos establecidos y funcionen correctamente. Las pruebas han sido diseñadas para identificar y corregir posibles defectos antes de que el software sea desplegado en un entorno de producción.

# pRUEBAS DE CAJA BLANCA

## Cobertura de Sentencias

**Descripción:**

* Verificar que todas las líneas de código son ejecutadas.

**Prueba:**

* Verificar que cada línea de código en la función “lista\_productos” es ejecutada al menos una vez.
* **Código de prueba:**

pythonCopiar códigoimport unittestfrom app import app  
class TestStatementCoverage(unittest.TestCase):  
  
    def setUp(self):  
        self.app = app.test\_client()        self.app.testing = Truedef test\_lista\_productos(self):  
        response = self.app.get('/')  
        self.assertEqual(response.status\_code, 200)  
 self.assertIn(b'Camisa', response.data)  
 self.assertIn(b'Pantalón', response.data)  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 unittest.main()

## Cobertura de Ramas

**Descripción:**

* Verificar que todas las ramas del código son ejecutadas.

**Prueba:**

* Verificar que todas las ramas en el código son ejecutadas, incluyendo el bucle que itera sobre “productos”.
* **Código de prueba:**

pythonCopiar códigoimport unittestfrom app import app  
class TestBranchCoverage(unittest.TestCase):  
  
    def setUp(self):  
        self.app = app.test\_client()        self.app.testing = Truedef test\_lista\_productos(self):  
        global productos  
        productos = [            {'id': 1, 'nombre': 'Camisa', 'descripcion': 'Camisa de algodón', 'foto': 'camisa.jpg'},  
            {'id': 2, 'nombre': 'Pantalón', 'descripcion': 'Pantalón de mezclilla', 'foto': 'pantalon.jpg'}  
        ]        response = self.app.get('/')  
        self.assertEqual(response.status\_code, 200)  
        self.assertIn(b'Camisa', response.data)  
        self.assertIn(b'Pantalón', response.data)  
  
 def test\_lista\_productos\_vacia(self):  
 global productos  
 productos = []  
 response = self.app.get('/')  
 self.assertEqual(response.status\_code, 200)  
 self.assertNotIn(b'Camisa', response.data)  
 self.assertNotIn(b'Pantalón', response.data)  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 unittest.main()

## Cobertura de DESiCIones

**Descripción:**

* Verificar que todas las condiciones posibles se prueban.

**Prueba:**

* Verificar que todas las condiciones posibles (si hay productos o no) se prueban.
* **Código de prueba:**

pythonCopiar códigoimport unittestfrom app import app  
class TestConditionCoverage(unittest.TestCase):  
  
    def setUp(self):  
        self.app = app.test\_client()        self.app.testing = Truedef test\_lista\_productos(self):  
        global productos  
        productos = [            {'id': 1, 'nombre': 'Camisa', 'descripcion': 'Camisa de algodón', 'foto': 'camisa.jpg'},  
            {'id': 2, 'nombre': 'Pantalón', 'descripcion': 'Pantalón de mezclilla', 'foto': 'pantalon.jpg'}  
        ]        response = self.app.get('/')  
        self.assertEqual(response.status\_code, 200)  
        self.assertIn(b'Camisa', response.data)  
        self.assertIn(b'Pantalón', response.data)  
  
 def test\_lista\_productos\_vacia(self):  
 global productos  
 productos = []  
 response = self.app.get('/')  
 self.assertEqual(response.status\_code, 200)  
 self.assertNotIn(b'Camisa', response.data)  
 self.assertNotIn(b'Pantalón', response.data)  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 unittest.main()

## Cobertura de Caminos

**Descripción:**

* Verificar que todos los caminos posibles en el código se ejecutan.

**Prueba:**

* Verificar que todos los caminos posibles en el código se ejecutan.
* **Código de prueba:**

pythonCopiar códigoimport unittestfrom app import app  
class TestPathCoverage(unittest.TestCase):  
  
    def setUp(self):  
        self.app = app.test\_client()        self.app.testing = Truedef test\_lista\_productos(self):  
        global productos  
        productos = [            {'id': 1, 'nombre': 'Camisa', 'descripcion': 'Camisa de algodón', 'foto': 'camisa.jpg'},  
            {'id': 2, 'nombre': 'Pantalón', 'descripcion': 'Pantalón de mezclilla', 'foto': 'pantalon.jpg'}  
        ]        response = self.app.get('/')  
        self.assertEqual(response.status\_code, 200)  
        self.assertIn(b'Camisa', response.data)  
        self.assertIn(b'Pantalón', response.data)  
  
 def test\_lista\_productos\_vacia(self):  
 global productos  
 productos = []  
 response = self.app.get('/')  
 self.assertEqual(response.status\_code, 200)  
 self.assertNotIn(b'Camisa', response.data)  
 self.assertNotIn(b'Pantalón', response.data)  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 unittest.main()

## Camino más corto

**Descripción:**

* Verificar que el camino más corto en el código se ejecuta correctamente.
* **Código de prueba:**

pythonCopiar códigoimport unittestfrom app import app  
class TestShortestPath(unittest.TestCase):  
  
    def setUp(self):  
        self.app = app.test\_client()        self.app.testing = Truedef test\_camino\_mas\_corto(self):  
        global productos  
        productos = []        response = self.app.get('/')  
        self.assertEqual(response.status\_code, 200)  
 self.assertNotIn(b'Camisa', response.data)  
 self.assertNotIn(b'Pantalón', response.data)  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 unittest.main()

# PRUEBAS DE CAJA NEGRA

## TRANSICIÓN DE ESTADO (SHOPPING):

Para realizar pruebas de caja negra específicamente para la transición de estados en el modelo Shopping, primero debemos considerar qué estados o transiciones estamos probando. Dado que el modelo Shopping en sí no tiene un campo de estado explícito, asumiremos que las pruebas están relacionadas con la transición del estado de las referencias (productos, variedades, clientes) asociadas con el Shopping, o bien con las transiciones del objeto Shopping mismo en términos de existencia y manejo.

### Escenarios y Estados

1. **Creación del Objeto Shopping**:
   * Transición del estado inicial (sin datos) a un estado creado (con datos válidos).
2. **Eliminación del Objeto Shopping**:
   * Transición del estado de existencia a no existencia.
3. **Actualización del Objeto Shopping**:
   * Aunque el modelo Shopping no tiene campos actualizables directamente en el esquema proporcionado, podríamos considerar cambios en los datos asociados a sus referencias (productos, variedades, clientes) como una forma indirecta de transición de estado.

## Pruebas de Valor Límite

Para las pruebas de caja negra de valores límites del modelo SalesDetail, nos enfocaremos en probar los límites de los campos numéricos y de cadena para asegurarnos de que el sistema maneje correctamente tanto los valores válidos como los inválidos.

### Valores Límites a Probar:

1. **Year, Month, Day**:
   * Valores válidos mínimos y máximos.
   * Valores fuera del rango esperado.
2. **Subtotal**:
   * Cadena vacía.
   * Cadena muy larga.
3. **Price\_unit, Amount**:
   * Valores válidos mínimos y máximos.
   * Valores fuera del rango esperado.

### Casos de Valor Límite (USUARIO)

* **Campo name**:
  + **Límite Inferior**: Prueba con una cadena vacía para asegurar que se valida la presencia del nombre.
  + **Límite Superior**: Prueba con una cadena muy larga para asegurar que el sistema maneja correctamente entradas extensas.
* **Campo lastName**:
  + **Límite Inferior**: Prueba con una cadena vacía para verificar que el sistema rechaza nombres de apellido vacíos.
  + **Límite Superior**: Prueba con una cadena muy larga para verificar el manejo de entradas extensas.
* **Campo email**:
  + **Límite Inferior**: Prueba con un formato de email inválido para asegurar que se validan correctamente los correos electrónicos.
  + **Límite Superior**: Prueba con un correo electrónico muy largo para verificar que el sistema maneja correctamente correos extensos.
* **Campo role**:
  + **Límite Inferior**: Prueba con una cadena vacía para asegurar que se valida la presencia del rol.
  + **Límite Superior**: Prueba con una cadena muy larga para verificar el manejo de entradas extensas.
* **Campo password**:
  + **Límite Inferior**: Prueba con una contraseña muy corta para asegurarte de que se valida correctamente la longitud mínima.
  + **Límite Superior**: Prueba con una contraseña extremadamente larga para verificar que el sistema maneja correctamente entradas extensas.
* **Campo state**:
  + Verifica que el campo puede ser true o false, asegurando que acepta ambos valores booleanos.

### Explicación de las Pruebas

* **Campo name**:
  + Prueba para una cadena vacía y una muy larga para verificar validación y manejo de longitud.
* **Campo lastName**:
  + Similar a name, pruebas con cadena vacía y muy larga.
* **Campo email**:
  + Pruebas con formato inválido, válido y correo muy largo para verificar validación de formato y longitud.
* **Campo role**:
  + Pruebas con cadena vacía y muy larga para verificar validación y manejo de longitud.
* **Campo password**:
  + Pruebas con una contraseña corta, válida y extremadamente larga para verificar validación y manejo de longitud.
* **Campo state**:
  + Verifica que el campo acepta valores booleanos true y false.

## Tablas de Decisión

Para realizar pruebas de caja negra usando la técnica de la **tabla de decisiones** en el modelo Sales, debemos identificar diferentes combinaciones de entradas y sus resultados esperados. La idea es cubrir distintos escenarios posibles que puedan surgir al usar el modelo, asegurando que el sistema maneje correctamente cada situación.

### Campos y Valores de Decisión:

* **transaction**:
  + Cadena válida
  + Cadena vacía
* **year**:
  + Año válido (por ejemplo, 2023)
  + Año fuera de rango (por ejemplo, 9999)
* **month**:
  + Mes válido (1-12)
  + Mes fuera de rango (0, 13)
* **day**:
  + Día válido (1-31)
  + Día fuera de rango (0, 32)
* **serie**:
  + Número válido (mayor a 0)
  + Número inválido (menor o igual a 0)
* **total**:
  + Número válido (mayor a 0)
  + Número inválido (menor o igual a 0)
* **sending**:
  + Número válido (mayor o igual a 0)
  + Número inválido (negativo)
* **state**:
  + Estado válido (por ejemplo, 'completed', 'pending')
  + Estado inválido
* **customer**:
  + Referencia válida (ID de cliente existente)
  + Referencia inválida (ID de cliente inexistente)
* **addres**:
  + Referencia válida (ID de dirección existente)
  + Referencia inválida (ID de dirección inexistente)

### Explicación de las Pruebas

1. **Creación del Objeto Shopping**:
   * Se prueba que un nuevo objeto Shopping pueda ser creado correctamente con datos válidos y que pase del estado no existente al estado existente.
2. **Eliminación del Objeto Shopping**:
   * Se verifica que un objeto Shopping pueda ser eliminado correctamente y que pase del estado existente al estado no existente.
3. **Actualización de Referencias**:
   * Aunque no se puede actualizar directamente el objeto Shopping, se pueden actualizar las referencias de product y variety, y verificar que estos cambios se reflejen correctamente en el objeto Shopping.

### Tabla de Decisión (VARIETY)

Para el modelo Variety, considera las siguientes condiciones y acciones:

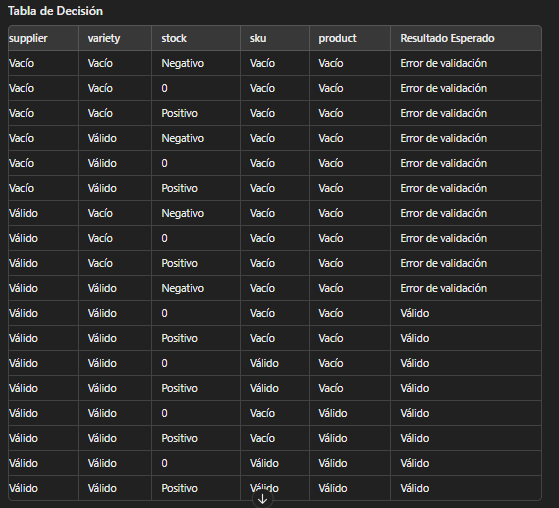
### Condiciones

* **Campo supplier**:
  + Puede estar vacío (Inválido)
  + Puede contener una cadena válida (Válido)
* **Campo variety**:
  + Puede estar vacío (Inválido)
  + Puede contener una cadena válida (Válido)
* **Campo stock**:
  + Puede ser un número negativo (Inválido)
  + Puede ser cero (Válido)
  + Puede ser un número positivo (Válido)
* **Campo sku**:
  + Puede estar vacío (Inválido)
  + Puede contener una cadena válida (Válido)
* **Campo product**:
  + Puede estar vacío (Inválido)
  + Puede ser una referencia válida (Válido)

### Acciones

1. **Validar que se pueden crear variedades con datos válidos.**
2. **Rechazar creaciones con datos inválidos.**

### TABLA DE DECISIÓN:



## Casos de Uso

Las pruebas de caja negra se enfocan en evaluar el funcionamiento del sistema en base a sus entradas y salidas sin considerar su estructura interna. Para el modelo Review, vamos a crear pruebas de caja negra que cubren los siguientes casos de uso:

1. **Creación de una reseña válida**: Crear una reseña con todos los campos válidos.
2. **Creación de una reseña con campos faltantes**: Intentar crear una reseña sin algunos campos obligatorios.
3. **Creación de una reseña con tipos de datos incorrectos**: Intentar crear una reseña con tipos de datos incorrectos para los campos.
4. **Consulta de reseñas**: Recuperar las reseñas de la base de datos.
5. **Actualización de una reseña**: Actualizar los campos de una reseña existente.
6. **Eliminación de una reseña**: Eliminar una reseña existente de la base de datos.

## Casos de Uso para el Modelo Subcategory

Para realizar pruebas de caja negra basadas en casos de uso para el modelo Subcategory, es importante identificar y definir los casos de uso que se relacionan con las operaciones CRUD (crear, leer, actualizar y eliminar) sobre las subcategorías.

1. **Crear una Subcategoría**:
   * Verificar que se puede crear una subcategoría con datos válidos.
   * Asegurar que no se puede crear una subcategoría sin un título o una categoría.
2. **Leer Subcategorías**:
   * Verificar que se pueden recuperar subcategorías por su ID.
   * Verificar que se pueden listar todas las subcategorías.
3. **Actualizar una Subcategoría**:
   * Verificar que se puede actualizar el título de una subcategoría.
   * Asegurar que no se puede actualizar con datos inválidos.
4. **Eliminar una Subcategoría**:
   * Verificar que se puede eliminar una subcategoría por su ID.
   * Asegurar que no se puede recuperar una subcategoría después de su eliminación.

### Explicación de las Pruebas

1. **Creación de Subcategoría**:
   * Verifica que la subcategoría se crea correctamente con un título y una categoría válidos.
   * Verifica que no se puede crear una subcategoría sin un título o una categoría, probando la validación de campos requeridos.
2. **Lectura de Subcategorías**:
   * Verifica que se puede recuperar una subcategoría por su ID.
   * Verifica que se puede listar todas las subcategorías existentes.
3. **Actualización de Subcategoría**:
   * Verifica que se puede actualizar el título de una subcategoría existente.
4. **Eliminación de Subcategoría**:
   * Verifica que una subcategoría puede ser eliminada por su ID y que ya no existe después de la eliminación.

# C:\Users\nyels\Downloads\WhatsApp Image 2024-08-06 at 9.05.42 PM.jpegANEXOS

