UNIVERSIDAD DE COSTA RICA

SEDE DE OCCIDENTE SO

IF6100 ANÁLISIS & DISEÑO DE SISTEMAS
EXAMEN PARCIAL 2
CARNÉ: C31621
NOMBRE: LUIS CAMPOS VARGAS

26 DE NOVIEMBRE DEL 2025
TOTAL DE PUNTOS: *100*

VALOR: 20 %

INDICACIONES

- Lea, con detenimiento, el contenido del examen.
- Puede efectuar consultas sólo en los primeros 45 minutos de la entrega de la prueba de parte del Docente a los estudiantes.
- Todas sus respuestas debe incorporarlas en el presente enunciado y subirlas en la sección de Mediación Virtual publicada en el entorno del curso para este examen.
- Puede utilizar las herramientas de software que hemos venido utilizando en el curso, para el proceso de análisis, diseño y construcción, pero los artefactos resultantes debe integrarlos en este documento.
- En el caso de anexar imágenes, si las mismas no son inteligibles no serán objeto de calificación.
- El examen es a cuaderno abierto y para realizar en la casa de habitación.
- Esta prueba es individual, por lo que cualquier infracción implicará anular la prueba de los involucrados y se procederá a aplicar las sanciones que indica el reglamento de la Universidad de Costa Rica
- La respuesta al examen debe entregarse a través de la plataforma de Mediación Virtual a más tardar el miércoles 26 de noviembre a la 05:00 P.M.
-

- SECCIÓN A (40 puntos)

Con base en el planteamiento y las descripciones de los casos de uso por favor proceda a desarrollar lo que se le solicita.

Caso de Sistema de Cita de pruebas de Manejo:

Suponga que la Dirección General de Tránsito le contrata como analista de sistemas para que desarrolle un nuevo sistema de información para la gestión de citas de pruebas de manejo.

■ **Las personas usuarias**, que estén registradas en el sistema, pueden **agendar citas para pruebas de manejo** (teórico o práctico).
■ El sistema deberá, de forma periódica (al menos una vez por semana), **verificar si existen citas pendientes con comprobantes de pago inválidos**. Si fuese así, se enviará, vía correo electrónico, **una notificación a la persona usuaria indicando el rechazo y los pasos para corregirlo**.
■ Una persona usuaria puede **cancelar una cita previamente agendada**. La cancelación debe registrarse en el sistema y liberar el espacio para que otra persona pueda reservarlo.
■ Para cada **tipo de examen** se deberá guardar:

- Nombre del examen (teórico o práctico)

- Duración
- Requisitos
- Código único del examen

■ Para cada **cita agendada** se deberá conservar:

- Fecha y hora
- Estado (pendiente, confirmada, cancelada, rechazada)
- Tipo de examen asociado
- Usuario que la solicitó

■ Para cada **comprobante de pago** se deberá almacenar:

- Número de comprobante
- Monto
- Fecha de pago
- Estado (válido, inválido)

■ Para las **personas usuarias** interesa conocer:

- Número de usuario
- tipoCedula(nacional/extranjero)
- Nombre
- Apellidos
- Dirección
- Teléfono
- Fecha de inscripción
- Correo electrónico

■ En el caso de la **validación del pago**, además de los datos del comprobante, debe registrarse:

- Fecha de validación
- Resultado (aprobado/rechazado)
- Mensaje de respuesta de la entidad bancaria

■ Para las **notificaciones**, el sistema debe enviar correos electrónicos en los siguientes casos:

- Confirmación de cita
- Rechazo por comprobante inválido
- Recordatorio de cita próxima

■ El sistema contará con 3 tipos de clientes:

- Una interfaz web para las personas usuarias (auto-registro y agendamiento)
- Una aplicación celular para los usuarios donde pueden validar la cita y agendar.
- Otra para los funcionarios encargados proceso en la organización separada de la web principal

A continuación, se le proveen dos casos de uso asociados a este sistema:

UC-001	**Agendar cita para prueba de manejo**

Actor(es)	**Principal: Persona Usuaria**	
Flujo de eventos básicos	**No.**	**Descripción del paso**
	1	El caso inicia cuando la persona usuaria ingresa su número de usuario y contraseña. El sistema valida credenciales.
	2	La persona selecciona el tipo de examen (práctico o teórico).
	3	El sistema muestra las fechas y horarios disponibles.
	4	La persona selecciona fecha y hora.
	5	El sistema solicita subir el comprobante de pago.
	6	El sistema valida el comprobante con la entidad bancaria.
	7	Si el comprobante es válido, el sistema agenda la cita y envía notificación por correo. Fin de caso de uso
Flujos alternos	**No.**	**Descripción del paso**
	1.1	En el paso 1, el sistema detecta que el número de usuario o contraseña es inválido. El caso de uso continúa en el paso 1.
	6.1	En el paso 6, el sistema detecta que el comprobante de pago no es válido. El caso de uso continúa en el paso 5.

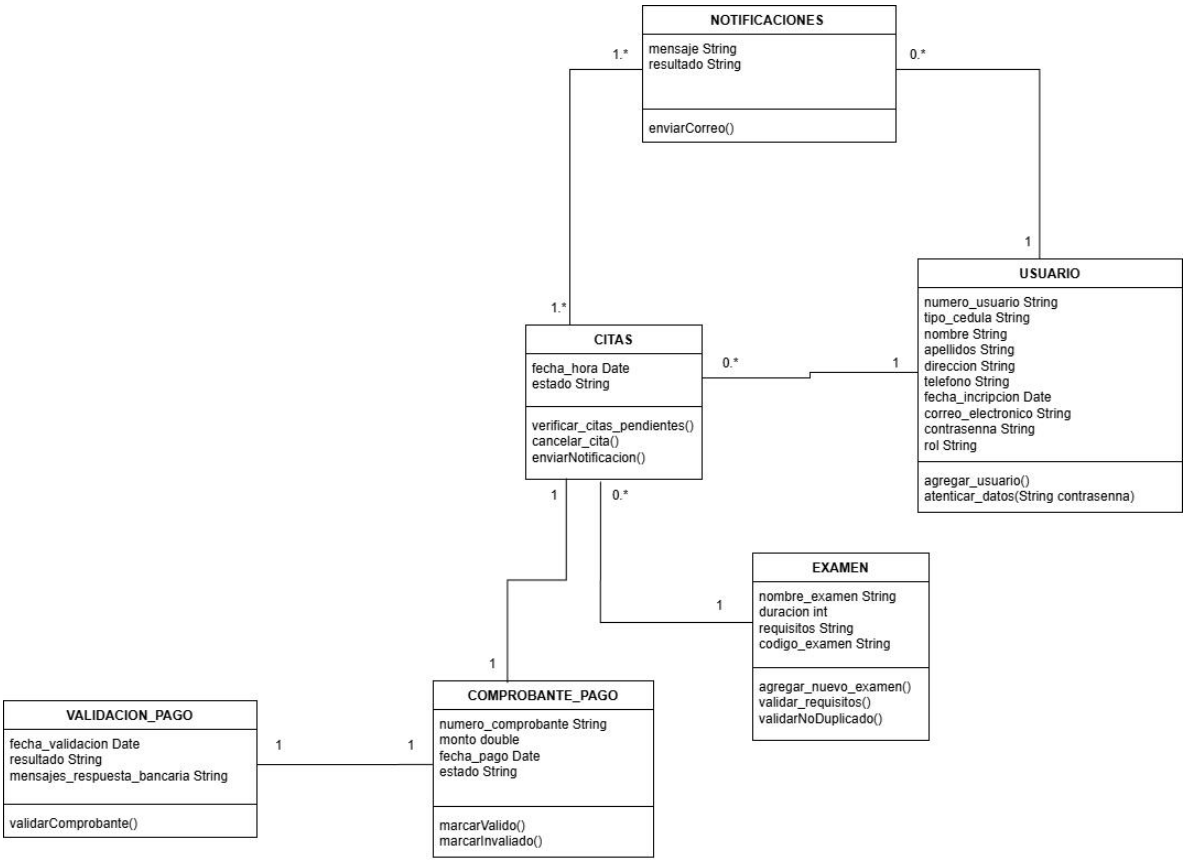
UC-002	**Registrar nuevo tipo de examen**	
Actor(es)	**Principal: Funcionario**	
Flujo de eventos básicos	**No.**	**Descripción del paso**
	1	El caso de uso comienza cuando el funcionario accede al sistema e ingresa sus credenciales (número de usuario y contraseña).
	2	El sistema valida la autenticidad de los datos ingresados y, si son

		correctos, concede acceso al menú principal.
	3	Una vez autenticado, el funcionario navega por el menú del sistema y selecciona la opción correspondiente para registrar un nuevo tipo de examen.
	4	El sistema responde mostrando la interfaz adecuada para esta operación.
	5	El sistema despliega un formulario en pantalla solicitando la información necesaria para el nuevo tipo de examen. Entre los campos requeridos se incluyen: • Nombre del examen (por ejemplo, “Prueba Teórica” o “Prueba Práctica”) • Duración estimada del examen • Requisitos previos que debe cumplir la persona usuaria para poder realizarlo
	6	El funcionario procede a completar todos los campos del formulario con la información correspondiente. Antes de guardar, revisa que los datos sean correctos y completos. Finalmente, confirma la operación presionando el botón “Guardar”.
	7	El sistema valida que la información ingresada no esté duplicada y cumple con los formatos requeridos. Si todo es correcto, almacena el nuevo tipo de examen en la base de datos y muestra un mensaje de confirmación indicando que el registro se realizó exitosamente. Fin de caso de uso.
Flujos alternos	**No.**	**Descripción del paso**
	2.1	En el paso 2, El sistema muestra mensaje: “Usuario o contraseña incorrectos”.
	2.2	Permite reintentar el inicio de sesión.
	2.3	Si falla 3 veces, bloquea temporalmente la cuenta y finaliza el caso.
	3.1	El sistema muestra mensaje: “No hay fechas disponibles para este examen”.
	3.2	Ofrece opción para seleccionar otro tipo de examen o salir.

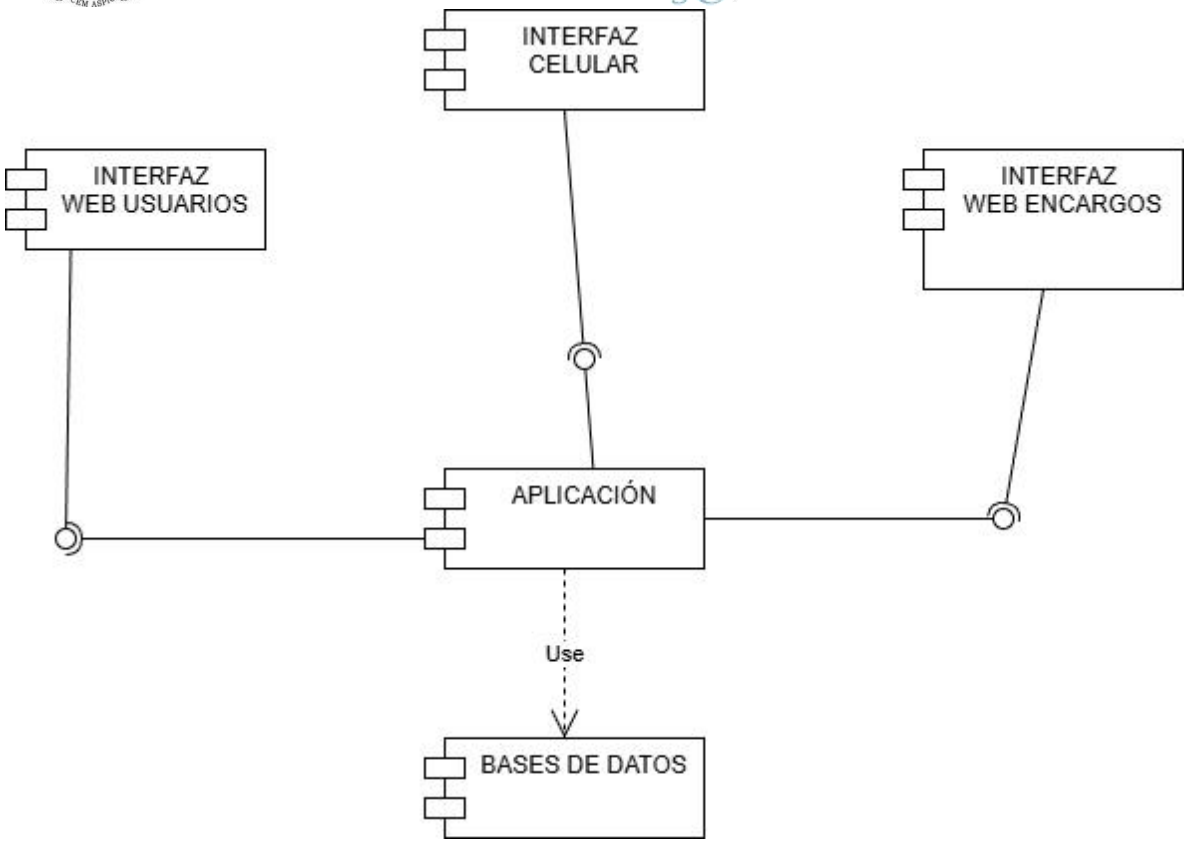
3.3	Si el usuario no selecciona otra opción, el caso finaliza.

1. **(30 puntos)** Desarrolle un <u>diagrama de clases</u> que muestre las clases relevantes, sus atributos y las asociaciones derivadas del caso del sistema para agendar citas de prueba de manejo y de los casos de uso asociados. El modelo propuesto será evaluado considerando que refleje correctamente las entidades de negocio involucradas, sus atributos, las asociaciones, roles, multiplicidades y direcciones de las relaciones.

Rubro	Puntaje
Entidades	**10**
Relaciones	**5**
Atributos	**10**
Operaciones	**5**



2. **(10 puntos)** Desarrolle un <u>diagrama de componentes</u> para la aplicación. Indique los supuestos que ha considerado para el desarrollo del diagrama solicitado.



Para elaborar el diagrama de componentes consideramos que el sistema los tres tipos de clientes independientes (la web para usuarios, la app móvil y la web para funcionarios), todos conectados a una única aplicación central que maneja la lógica del negocio y el acceso a una base de datos común. También se supuso que cada interfaz se comunica con la aplicación mediante servicios o APIs, que no existe comunicación directa entre las interfaces, que la seguridad y los permisos son gestionados por el backend.

SECCIÓN B (25 puntos)

Se requiere una aplicación en **Java** que permita ingresar la temperatura actual de una máquina y mostrar una alerta dependiendo del rango en el que se encuentre. La aplicación debe solicitar:

- **Temperatura actual (en grados Celsius)**

Descripción de la necesidad

Verificar Temperatura devuelve un número:

- -1 → Error fuera de rango
- 0 → Bajo cero
- 1 → Normal
- 2 → Precaución
- 3 → Crítico
- 4 → Peligro

Clase a probar:

```
package cr.go.ucr.examen1;


public class MonitoreoTemperatura {

   public static int verificarTemperatura(Double temperatura) {
      if (temperatura == null || temperatura < -50.0 || temperatura > 200.0) {
         return -1; // Código de error
      }
      if (temperatura < 0) {
         return 0;
      } else if (temperatura <= 50) {
         return 1;
      } else if (temperatura <= 80) {
         return 2;
      } else if (temperatura <= 100) {
         return 3;
      } else {
         return 4;
      }
   }

}
```

(10 puntos) Mencione los casos de pruebas de caja negra. Utilice el particionamiento equivalente, e identifique la condición de entrada, clases válidas y clases inválidas.

- Partición equivalente
  Clases Válidas:
  Condición: Temp < 0
  Condición: 0 < Temp ≤ 50.0
  Condición: 50.0 < Temp ≤ 80.0
  Condición: 80.0 < Temp ≤ 100.0
  Condición: Temp > 100

Clases Inválidas

temp < -50.0

temp > 200.0

temp = null

- Análisis valores limites

  -50.1,-50.0, -49.9

  -0.1, 0.0 , 0.1

  49.9, 50.0 ,50.1

  79.9, 80.0 , 80.1

  99.9, 100.0 , 100.1

  199.9, 200.0 , 200.1

(**15 puntos**) Realizar las pruebas de caja negra identificadas en el punto anterior utilizando Junit Test como fue visto en clases. Genere las evidencias de la ejecución de las pruebas y las entrega, así como el código fuente utilizado.

```
package com.mycompany.examen_2_pruebas;

import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.*;

/**
 *
 * @author varga
 */
public class MonitoreoTemperaturaTest {

    // ============================================================
    //                  PARTICIÓN EQUIVALENTE
    // ============================================================
    @Test
    public void testPE_TemperaturaNull() {
        int expResult = -1;
        int result = MonitoreoTemperatura.verificarTemperatura(null);
        assertEquals(expResult, result);
    }
```

```
@Test
public void testPE_TemperaturaMenorRango() {
   int expResult = -1;
   int result = MonitoreoTemperatura.verificarTemperatura(-60.0);
   assertEquals(expResult, result);
}

@Test
public void testPE_TemperaturaMayorRango() {
   int expResult = -1;
   int result = MonitoreoTemperatura.verificarTemperatura(250.0);
   assertEquals(expResult, result);
}

@Test
public void testPE_BajoCero() {
   int expResult = 0;
   int result = MonitoreoTemperatura.verificarTemperatura(-10.0);
   assertEquals(expResult, result);
}

@Test
public void testPE_Normal() {
   int expResult = 1;
   int result = MonitoreoTemperatura.verificarTemperatura(25.0);
   assertEquals(expResult, result);
}

@Test
public void testPE_Precaucion() {
   int expResult = 2;
   int result = MonitoreoTemperatura.verificarTemperatura(70.0);
   assertEquals(expResult, result);
}

@Test
public void testPE_Critico() {
   int expResult = 3;
   int result = MonitoreoTemperatura.verificarTemperatura(90.0);
   assertEquals(expResult, result);
}

@Test
```

```
public void testPE_Peligro() {
   int expResult = 4;
   int result = MonitoreoTemperatura.verificarTemperatura(150.0);
   assertEquals(expResult, result);
}

// ============================================================
//              ANÁLISIS DE VALORES LÍMITE (AVL)
// ============================================================
// ----------- Límite: -50 -----------
@Test
public void testAVL_Menos50_1() {
   int expResult = -1;
   int result = MonitoreoTemperatura.verificarTemperatura(-50.1);
   assertEquals(expResult, result);
}

@Test
public void testAVL_Menos50_0() {
   int expResult = 0;
   int result = MonitoreoTemperatura.verificarTemperatura(-50.0);
   assertEquals(expResult, result);
}

@Test
public void testAVL_Menos50_2() {
   int expResult = 0;
   int result = MonitoreoTemperatura.verificarTemperatura(-49.9);
   assertEquals(expResult, result);
}

// ----------- Límite: 0 -----------
@Test
public void testAVL_0_1() {
   int expResult = 0;
   int result = MonitoreoTemperatura.verificarTemperatura(-0.1);
   assertEquals(expResult, result);
}

@Test
public void testAVL_0_0() {
   int expResult = 1;
   int result = MonitoreoTemperatura.verificarTemperatura(0.0);
```

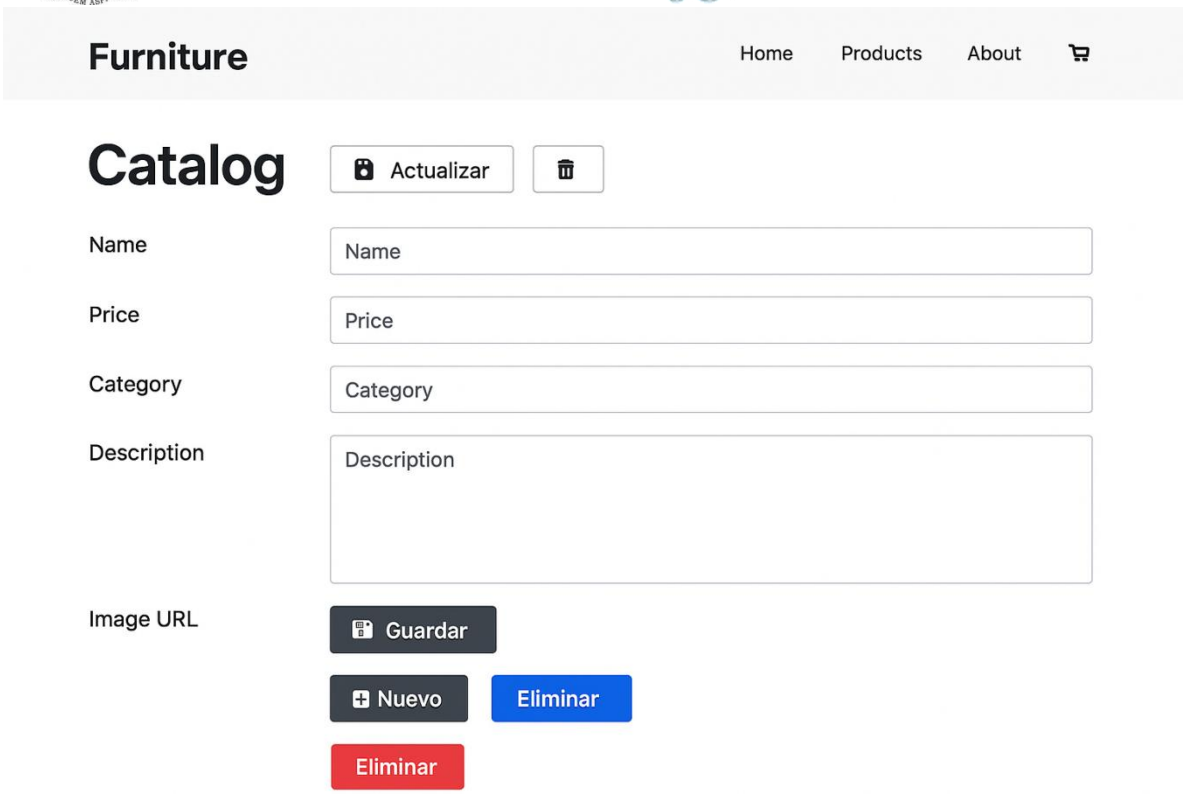
```
    assertEquals(expResult, result);
}

@Test
public void testAVL_0_2() {
    int expResult = 1;
    int result = MonitoreoTemperatura.verificarTemperatura(0.1);
    assertEquals(expResult, result);
}

// ---------- Límite: 50 ----------
@Test
public void testAVL_50_1() {
    int expResult = 1;
    int result = MonitoreoTemperatura.verificarTemperatura(49.9);
    assertEquals(expResult, result);
}

@Test
public void testAVL_50_0() {
    System.out.println("AVL: 50.0");
    int expResult = 1;
    int result = MonitoreoTemperatura.verificarTemperatura(50.0);
    assertEquals(expResult, result);
}

@Test
public void testAVL_50_2() {
    int expResult = 2;
    int result = MonitoreoTemperatura.verificarTemperatura(50.1);
    assertEquals(expResult, result);
}

// ---------- Límite: 80 ----------
@Test
public void testAVL_80_1() {
    int expResult = 2;
    int result = MonitoreoTemperatura.verificarTemperatura(79.9);
    assertEquals(expResult, result);
}

@Test
public void testAVL_80_0() {
```

```
    int expResult = 2;
    int result = MonitoreoTemperatura.verificarTemperatura(80.0);
    assertEquals(expResult, result);
}

@Test
public void testAVL_80_2() {
    int expResult = 3;
    int result = MonitoreoTemperatura.verificarTemperatura(80.1);
    assertEquals(expResult, result);
}

// ---------- Límite: 100 ----------
@Test
public void testAVL_100_1() {
    int expResult = 3;
    int result = MonitoreoTemperatura.verificarTemperatura(99.9);
    assertEquals(expResult, result);
}

@Test
public void testAVL_100_0() {
    int expResult = 3;
    int result = MonitoreoTemperatura.verificarTemperatura(100.0);
    assertEquals(expResult, result);
}

@Test
public void testAVL_100_2() {
    int expResult = 4;
    int result = MonitoreoTemperatura.verificarTemperatura(100.1);
    assertEquals(expResult, result);
}

// ---------- Límite: 200 ----------
@Test
public void testAVL_200_1() {
    int expResult = 4;
    int result = MonitoreoTemperatura.verificarTemperatura(199.9);
    assertEquals(expResult, result);
}

@Test
```

```
  public void testAVL_200_0() {
     int expResult = 4;
     int result = MonitoreoTemperatura.verificarTemperatura(200.0);
     assertEquals(expResult, result);
  }

  @Test
  public void testAVL_200_2() {
     int expResult = -1;
     int result = MonitoreoTemperatura.verificarTemperatura(200.1);
     assertEquals(expResult, result);
  }
}
```

SECCIÓN C (15 puntos)

A continuación, se presenta el siguiente caso:
Una empresa crea una página de una mueblería para ventas de sus productos. En la cual, aunque se dio una plantilla base, al repartirse las tareas entre distintos programadores se tuvieron ciertas inconsistencias al crear la la aplicación entre ellas:
-Se utilizaron distintos iconos para las acciones de guardado, nuevo y eliminar en la parte de creación del catalogo en la trastienda. Además, los botones para ejecutar las acciones de guardado de información en los catálogos salen etiquetas que dicen guardar otras sale como actualizar, borrar, eliminar entre otros.
-Existen acciones como borrar o pagar que se ejecutan inmediatamente y producen muchas equivocaciones por parte de los usuarios, de las cuales ha habido muchas quejas porque está muy sensible el botón o no fueron ellos. Además, es muy difícil recuperar los borrados de información errónea o actualizaciones de datos.
-La imágenes de productos en ocasiones no aparece y muestra un error en la pantalla.
-Los mensajes de error pueden salir en una esquina de la pantalla o al final de la pantalla por lo que los usuarios se quejan de que no ven los mensajes de error. Esto ocasiona que haya muchos problemas a la hora de saber cual es el error en la digitación de los datos.
-Los clientes de compra pierden el carrito de compra muy seguido a menos que les den comprar, además, aunque algunos hacen compras muy recurrentes, tienen que digitar siempre lo mismo.



(15 puntos) según el caso mencionado anteriormente y Mencione 7 problemas al diseñar la interface y cuales consejos que deben utilizarse para mejorar las mismas.

Problema	Consejo
1. Inconsistencia en iconos y etiquetas de botones	Mantener un diseño común para todas las interfaces (Template). Proveer acciones, comandos y respuestas consistentes y predecibles (semejanza entre pantallas).
2. Acciones sensibles sin confirmación	Pedir confirmación ante borrados y permitir métodos de recuperación.
3. Mensajes de error poco visibles	Poner los errores de manera visible. Hacer los mensajes lo más específicos posibles (ejemplo: “El valor debe estar entre 1 y 5”).
4. Imágenes de productos no cargan	Validar la ruta de las imágenes. Mostrar una imagen por defecto si la original no carga.
5. Pérdida del carrito de compras	Mantener el carrito siempre visible en pantalla (ícono o panel fijo) y guardar los datos en sesión o almacenamiento local

	para que no se pierdan aunque el usuario cierre el navegador.
6. Usuarios recurrentes deben ingresar datos repetitivos	Usar autollenado si es posible. Mostrar valores por defecto cuando sea posible. No pedir información que se puede obtener del sistema
7. Diseño poco atractivo y sobrecargado	Usar colores apropiados y consistentes, evitar saturación visual. Dejar espacios despejados y agrupar información relacionada.

SECCIÓN D (20 puntos)

(5 puntos) Explique cuáles son los roles principales en Scrum, cuál es la responsabilidad de cada uno y cómo interactúan durante el desarrollo del producto.

Product Owner (Propietario del Producto): Es la persona que tiene clara la idea de lo que se quiere construir. Se encarga de definir la visión del producto y de decidir qué cosas son más importantes en la lista de tareas (el backlog). Está en contacto directo con el equipo y con el Scrum Master para asegurarse de que el trabajo vaya en la dirección correcta y que lo que se entrega realmente sirva al negocio.

Scrum Master: Es como el facilitador del equipo. Su trabajo es que el proceso Scrum se cumpla bien, que no haya obstáculos y que todos puedan comunicarse de forma fluida. También ayuda a que las reuniones de planificación y los sprints se hagan de manera ordenada, y actúa como un puente entre el Product Owner y el equipo de desarrollo.

Equipo de Desarrollo: Son las personas que hacen el trabajo técnico. Es un grupo pequeño y variado que se organiza por sí mismo para convertir las tareas en un producto funcional. Ellos colaboran con el Product Owner para entender qué es lo más importante y con el Scrum Master para mejorar la forma en que trabajan. Lo hacen en ciclos cortos llamados sprints, y al final de cada uno entregan algo que ya funciona.

(5 puntos) Explique cómo Kanban muestra el progreso de las tareas se mueven dentro de esta metodología y combinada con scrum quien se encarga de validar la revisión final del producto.

Kanban muestra el progreso de las tareas mediante una tabla con columnas (pila de requerimientos, planificadas, en desarrollo, pruebas y verificadas), donde las tareas se mueven de una columna a otra según su estado. Combinado con Scrum, la validación final del producto se realiza en la reunión de revisión de sprint, donde el Product Owner, junto

con el equipo y las partes interesadas, decide si el incremento entregado se acepta como completo.

(5 puntos) Como se define una prueba de caja blanca y cuáles son los criterios que definen que la hicimos correctamente.

Una prueba de caja blanca se define como aquella que utiliza la estructura interna del código para diseñar casos de prueba, siguiendo el flujo de ejecución y midiendo qué partes del programa se recorren. Se considera correcta cuando cumple criterios de cobertura como: ejecutar todas las sentencias, evaluar cada decisión en sus dos resultados, probar cada condición en verdadero y falso, y en casos más avanzados, cubrir todas las combinaciones de condiciones posibles.

(5 puntos) Para diseño de sistemas que es Acoplamiento y Cohesión.

En diseño de sistemas, el acoplamiento es el nivel de dependencia entre módulos, y se busca que sea bajo para que los cambios no afecten a todo el sistema. La cohesión es el grado en que las funciones de un módulo están relacionadas y trabajan juntas, y se busca que sea alta para que cada módulo tenga una responsabilidad clara. El diseño ideal combina alta cohesión con bajo acoplamiento.