Índice

Consideraciones sobre el Der----------**2**

Consideraciones sobre el la implementación----------**3**

Clase Conexión----------**3**

Clase Gestor de Errores----------**3**

Clase Utilidades----------**3**

Roles----------**3**

Clientes/Proveedores y Usuarios----------**4**

Consideraciones sobre el Administrador----------**4**

Consideraciones sobre los procedures----------**4**

Índices----------**4**

Estrategia de resolución

Consideraciones sobre el DER

- Consideramos que cada compra de una oferta imprime un cupón, por ende si quiero comprar n ofertas, voy a tener que realizar n compras y comprar de a 1 cupón.
- De esto se desprende la siguiente aclaración: El límite de compra por usuario hace referencia a la cantidad de veces que puede comprar un cliente dado. Esto tiene mucho más sentido desde mi punto de vista que hacer un límite de compra por usuario por transacción. Es decir supongamos que tengo como límite_de_compra=3:
  a) Partiendo de la base de que el límite es por la cantidad de veces que lo compra un usuario en total, yo podría solo comprar 3 veces esa oferta. A partir de la 4ta, habría un error, informando de que ya compré demasiadas.
  b) Partiendo de que el límite es por transacción, yo como cliente podría comprar de a 3 ofertas hasta agotar todo el stock y así quedarme con todos los cupones disponibles. Visto de ésta forma, si la idea del límite es limitar la cantidad de compras que puede hacer un cliente dado, no le veo sentido a que el límite sea por transacción ya que al fin y al cabo podría como cliente terminar comprando todo. Además ésta forma implicaría un cambio brusco en nuestro modelo, que ya fue encarado por el primer camino.

Consideramos que esa es la decisión más relevante a destacar del Der.

- Otro detalle a destacar es que cuando se crea un nuevo cliente, en la tabla agregamos un default de 200 como saldo de bienvenida, para que todo cliente creado empiece con ese dinero.

Consideraciones sobre la implementación

- Contamos con una **Clase Conexión** donde allí depositamos todos los métodos relacionados con la conexión a la Base de Datos, incluyendo todos los métodos para ejecutar procedures y recibir consultas.
- También desarrollamos una **Clase Gestor de errores**, donde creamos muchos métodos de verificación y en caso de que se cumpla tal condición, se lanzaba una excepción con un mensaje descriptivo del problema.
- **En la clase Utilidades**, se encuentran los métodos para obtener el hash de una contraseña ingresada, también las condiciones para los Listados con filtros de texto libre y exacto, y la **fecha que se obtiene del App.config.**
- **Algo importante a destacar: Roles y su diseño**: el enunciado decía que en un futuro (no ahora) un usuario iba a poder tener varios roles asignados. Dicho esto, no implementamos explícitamente unos formularios para poder asignarle un nuevo rol a un usuario existente. Pero por cómo esta modelado, nos permite tener usuarios con varios roles. Como prueba, cargamos también en la carga inicial un usuario 'user3' que tiene como rol Cliente y Proveedor. Para que funcione, lo diseñamos de la siguiente manera:
  Cuando un usuario se logea correctamente, le aparecerá un formulario preguntando con qué rol ingresar. Inicialmente cualquier persona que es nueva en el sistema sólo va a poder tener un solo rol, debido a que no desarrollamos esa funcionalidad de agregarle otro rol más, pero sí lo hicimos lo suficientemente extensible para que pueda cumplir en un futuro ese requerimiento. Y si se logea con el 'user3' pass:1234, podrá ingresar como Cliente o como Proveedor.

  Si se da el caso de que un administrador da de alta a nuevos roles, estos aparecerán también como disponibles en la lista de Roles disponibles al registrar un usuario. Aparecen todos los roles disponibles en el sistema menos el de administrador.

También al intentar ingresar con un rol dado, sólo se mostrarán los que estén habilitados.

- **Otra cosa importante a destacar: Clientes/Proveedores y su respectivo Usuario**: Debido a que inicialmente en el sistema va a haber muchos clientes y proveedores sin usuarios, permitimos desde el Administrador dar de alta a clientes y proveedores sin asignarle un nombre de usuario. Si se da el caso de que quiere crear un cliente nuevo con un usuario, deberá ir a la funcionalidad Registrar Usuario, donde se ingresa el username, rol deseado y según éste, se le abre un formulario u otro(Alta de Cliente, Alta de Proveedor)
- **Consideraciones sobre el usuario Administrador:** No le asignamos la funcionalidad "Comprar Oferta" al Administrador debido a que éste no va a tener un saldo, por lo tanto no le encontré sentido agregársela, pero sí posee todas las demás.

- **Con respecto a la ejecución de los procedures**, aquellos que devuelven un int para saber su valor de verdad y de acuerdo a ello, lanzar excepción o no, establecimos que devuelven 1 si es verdadero, y (-1) si es falso. Ejemplo: el procedure usuario_existente, que es ejecutado por el método verificarExistenciaDeUsuario, lanza una excepción si el valor recibido por el proc es 1, es decir, si ya existe un usuario con el nombre que le pasa por parámetro.

- **Creamos un índice** en la tabla Clientes y Proveedores para optimizar las búsquedas de registros por nombre de usuario, pero no los creamos en la tabla de Ofertas o Cupones porque consideramos que esas tablas van a tener mucho movimiento(muchos inserts) porque no imagino un escenario donde todo el tiempo ingresen nuevos usuarios al sistema constantemente, sí imagino que los proveedores existentes estén publicando muchas ofertas y los clientes, comprando, por lo tanto, las sentencias de insert/delete van a ser menos eficiente porque en cada modificación que sufra la tabla, se va a modificar también el índice (ya que el árbol B+ que está detrás sufrirá cambios, si por ejemplo un nodo intermedio llega a su porcentaje máximo de llenado y debe realizar un Split).