Sistema de Gestion SoftBoos (más Tienda Virtual)

Se plantea crear un sistema de gestion para una tiendas que sea ajustable sin importar que tipo de productos vendan. Dicho sistema se creara con php javascript html bootstrap 4.3 jquery mysql. Contara con la posibilidad de manejar (altas, bajas, modificaciones, o sea un CRUD) clientes, empleados, proveedores, productos, caja (movimientos compras, ventas, depositos, retiro, gostos, etc), compras, ventas, informes de operaciones y un sistema de audiotiras que controlara las acciones del usuario (empleado) sin excepciones.

El procedimiento para poder concretar el sistema es: desarrollarlo con el sistema modelo-vista-controlador, por el lado de la base de datos esta sera gestionada por medio de procedimientos almacenados, funciones y desencadenadores.

Herramientas usadas en el desarrollo

Editor de texto: visual studio code, sublime text

Software para el servidor local: EasyPHP 17, este contiene los siguientes elementos

Servidor: apache 2.4.25 x86

Motor de base de datos: mysql 5.7.17 x86

Gestor de base de datos: phpmyadmin 4.7.0

Lenguaje del backend: php 7.1.3 x86

Tecnologia para el frontend: html5, javascript, bootstrap 4.3, vali admin, y otras librerias en js.

Descripcion del sistema

El sistema contara con roles, siendo clientes y empleados los mas importantes, ya que estos son los que accederan al sistema y/o tienda virtual.

El cliente podra registrarse en la tienda, modificar sus datos, hacer pedidos, cancelar pediddos, gestionar su lista de deseos (agregar, quitar productos), suscribirse a la seccion de noticias (no hace falta estar registrado como cliente, solo proveer un correo electronico válido).

Los empleados podran cargar, actualizar y borrar clientes, otros empleados, cargos (roles de los empleados), productos ( para la venta/compra, insumos, recetas, combos), unidades de medida, categorias (productos), comprobantes de compra y venta (solo cargar), proveedores, relacionar proveedores con prodcutos, caja (registrar retiros o ingresos de plata provenientes de otras fuentes que no sean una compra o una venta), etc. Cabe destacar que al aplicar el sistema de roles el acceso a estos modulos puede limitar las funciones o acceso al mismo.

Modulos

Requisitos de los modulos:

* Un modulo puede tener altas, bajas y modificaciones
* Cada modulo se meneja con la arquitectura modelo-vista-controlador
* La capa de modelo o datos se conecta a la base de datos y le hace consultas a travez de procedimientos almacenados en la base de datos
* Los procedimientos y/o funciones almacenadas se encargan de gestionar consultas de tipo dml y ddl, como ser select, insert, update, delecte y create
* Cada tabla de la base de datos tendra detrás una tabla “auditoria” (según se necesite) que registrara cualquier accion (altas, bajas, modificaciones) de cualquier usuario (empleados y clientes – quiza de clientes no haga falta) en el sistema
* Las tablas de auditoria seran accesibles por procedimientos almacenados para recuperar sus registros (solo ver). Y triggers o disparadores para manipular los insert en dichas tablas
* Las tablas de auditoria tiene la siguiente estructura:
  + Un identificador “[nombretabla]\_id”
  + La estructura de la tabla que se quiere auditar, sin el/los atributo/s fecha\_alta, fecha\_baja
  + Un estado “estado\_id” (este es distinto del estado de la tabla que se esta auditando)
  + Una fecha (distinta a la fecha de alta de la tabla auditada)
  + El user name del “empleado” conectado a la base de datos
* Al crear un nuevo usuario empleado, tambien se generara un usuario (user) en la base de datos con la sentencia create dentro de un procedimiento almacenado que tendra solamente permisos de execute
* Formato para los disparadores: Tr{nombre de tabla}{select|insert|update}{que hace el disparador}[campo|atributo]
* Formato para los procedimientos almacenados: Sp|Sf{nombre de la tabla}{select|insert|update}{que hace el procedimiento/funcion almacenada}[campo|atributo]
* Agregar el autocomplete a los campos de los formularios que hagan falta, cuando el sistema pase a etapa de produccion.

Personas:

|  |  |  |
| --- | --- | --- |
| Cargos | Este modulo es un sinonimo de roles de usuarios pero solo a nivel de empleados del sistema (ej.: gerente, asistente, cajero, etc). Permite crear, modificar y eliminar cargos (solo por usuarios autorizados – administrador del sistema/dueño), da acceso a un cargo determinado de ver, agregar, modificar y eliminar en un modulo determinado. | |
| Hecho | * Alta de cargos * Baja (logica) de cargos * Modificacion de cargos * Asignar permisos a modulos * Creada la tabla de auditoria “cargos\_auditoria” * Disparador que establece la fecha de alta antes del insert * Disparador que inserta en la tabla de auditorias despues de un insert * Disparador que modifica en la tabla de auditorias despues de un update (se aplica a bajas logicas) * Disparador que inserta en la tabla de permisos los modulos existentes en la tabla modulos y carga los permisos por defecto desabilitados (sin permisos para acceder a los modulos) cada vez que se hace un insert de un nuevo cargo. * Se agrego un procedimiento almacenado que trae todos los registros de la tabla cargos, SpCargosSelectFull * Se agrego un procedimiento almacenado que trae todos los registros de la tabla cargos que esten activos e inactivos solamente, SpCargosSelectFullActivoInactivo * Se agrego un procedimiento almacenado que trae un solo cargo según los parametros id del cargo e id de estado (1 a 5, valores usados 3: para activos e inactivos o 4 para traer todos), SpCargosSelectOne * Se agrego validacion de datos en el controlador (capa de negocios) en altas, edicion y borrado * Se agrego controlar que no se repita el nombre de un cargo (alta y modificacion, debe ser desde en backend y tambien “puede” ser en el frontend) (hecho) * Se agrego el procedimiento almacenado para altas “SpCargosInsert”, para modificaciones “SpCargosUpdate”, para bajas “SpCargosDeleteLogic”. * Se agregaron los procedimientos almacenados para las siguientes funciones de cargoModel: selectNivelesAcceso() = “SpNivelAccesoSelect”, selectPermisos() = “SpPermisosSelect”, updatePermisos() = “SpPermisosUpdate”, getModulos() = “SpModulosSelectAll” |
| Agregar | * Agregar la llamada a los procediminetos almacenados desde la capa de datos (modelo) junto con el manejo de excepciones |
| Modificar | * Cambiar la capa de modelo de datos para llamar a las funciones y/o procedimientos almacenados en lugar de las consultas (query) |
| Empelados | Este modulo se encarga del manejo de los empleados, altas, modificaciones y bajas. | |
| Hecho |  |
| Agregar | * Verificar que al agregar un empleado no sea de un cargo mayor al del usuario actual que lo intenta dar de alta |
| Modificar | * Cambiar la capa de modelo de datos para llamar a las funciones y/o procedimientos almacenados en lugar de las consultas (query) |
| Clientes | Este modulo se encarga de almacenar los datos de los clientes del negocio en cuestion. Se cargan sus datos personales como ser DNI, nombre y apellido, tambien se penso en cargar otros datos de carácter mas personal como fecha de nacimiento, direccion, email, telefono, estos ultimos pueden no ser de importancia por lo que pueden ser ignorados (según los requisitos del comercio) | |
| Hecho | * Alta de clientes * Baja (logica) de clientes * Modificaiones de clientes * Ver los datos completos de los clientes * Se agrego la tabla de auditorias “clientes\_auditoria” * Se agrego triggers/disparadores que interactuan con la tabla de auditoria “clientes\_auditoria” al momento de actualizar/borrar (borrado logico) y agregar un registro en la tabla clientes. * Se agregaron los siguientes procedimientos almacenados: SpClientesSelectAll recibe un parametro si es 1 devuelve todos los clientes, si el parametro es mayor que 1 trae todos los clientes incluso los que fueron dados de baja; |
| Agregar | * Filtro avanzado que permita buscar clientes por otros campos no listados en el datatable (opcional según la solicitud del negocio) * Agregar procedimientos almacenados para hacer altas, bajas y modificaciones * Agregar que exista un control para DNI duplicados |
| Modificar | * Cambiar la capa de modelo de datos para llamar a las funciones y/o procedimientos almacenados en lugar de las consultas (query) * Modificar el formulario de altas y modificaciones poniendo los campos obligatorios primero y luego los demas (modificacion relcionada con agregar un control para DNI unico) |
| Proveedores |  | |
| Hecho |  |
| Agregar |  |
| Modificar | * Cambiar la capa de modelo de datos para llamar a las funciones y/o procedimientos almacenados en lugar de las consultas (query) |
| Sucursales |  | |
| Hecho |  |
| Agregar |  |
| Modificar | * Cambiar la capa de modelo de datos para llamar a las funciones y/o procedimientos almacenados en lugar de las consultas (query) |

Inventario:

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

Informes:

Auditorias:

Tienda Virtual:

Tesoreria:

Ventas:

Compras:

Nota: al crear el script de la base de datos debo tener en cuenta el orden de los insert en las tablas para que no salten errores por falta de datos ej.: insertar un producto que necesite de que exista el rubro/categoria primero. Separar la estructura de la base de datos de los registros (insert) de las tablas base (ej.: estado, forma de pago, y otras que no dependen de otra tabla).