Gestión de Datos

08-2030

UberFRBA

Estrategia

UTN.BA

UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

Profesor	REINOSA, Enrique	
Alumnos	RODRIGUEZ, Luis	149567-7
	BULBULIAN, Juan Pablo	116134-9
	COHEN, Damián	143668-5
	BROWARNIK, Gabriel	151893-8
Curso	K-3012	
Año	2017	
Cuatrimestre	1º	

Índice

Conexión con la Base de Datos

En la carpeta ConexiónBD se encuentran dos clases.

BDHandler.cs

Aquí se levanta la conexión desde el archivo App.config, el cual tiene una entrada llamada BDConfig con los datos para la conexión que exige el enunciado.

Posee tres métodos...

```
publicDataTableexecSelectSP(String nameStoredProcedure,
List<BDParametro>listParametros = null)
```

Permite ejecutar un SP que devuelva un Select con su lista de parámetros y nos devuelve un DataTable para mostrar en la aplicación.

```
public void execSP(String nameStoredProcedure, ref
List<BDParametro>listParametros)
```

Permite ejecutar un SP, con su lista de parámetros pasada por referencia, en la cual devuelve los parámetros de salida devueltos por el motor.

```
public List<String>execListSP(String nameStoredProcedure)
```

Permite ejecutar un SP, y devuelve una lista con la primera columna del Select que devuelve el motor.

BDParametro.cs

Sirve para el modelado de los parámetros de entrada y salida usados por los métodos de la clase **BDHandler.cs.**

Usuario

Se modelo al usuario en la clase.

Usuario.cs

La cual entiende los siguientes métodos.

`publicStringiniciarSesion()`

Utiliza el SP **`LJDG.iniciar_sesion`** para validar los datos de inicio de sesión. La contraseña ya viaja encriptada con el algoritmo SHA256.

`publicList<String>obtenerRolesUsuario()`

Utiliza el SP **`LJDG.obtener_roles_usuario`** para tener al usuario como variable global del sistema luego del login. Y devuelve una lista que será usada en el combobox en la segunda pantalla del login.

Login

Se cuenta con dos formularios dentro de la carpeta Logueo.

Login.cs

Aquí se solicita el nombre de usuario y contraseña la cual es encriptada al momento y a través de la clase Usuario.cs se validan los datos. Si el usuario no tiene roles asignados no deja ingresar a la aplicación.

SeleccionRol.cs

Aquí se muestra un combo desplegable con todos los roles disponibles para el usuario. Los cuales fueron obtenidos en el paso anterior a través de los métodos de la clase Usuario.cs. Una vez realizado un login exitoso se accede al menú principal de la aplicación.

Búsqueda de Chofer / Cliente

Se utiliza el formulario

BuscarIndividuo.cs.

```
publicBuscarIndividuo(Form _formPadre, String _tipoIndividuo, char _modo)
```

En **_formPadre** recibe el formulario anterior al cual le serán devueltos los parámetros del individuo a buscar. Estos son ID, nombre y apellido.

En **_tipoIndividuo** se especifica el tipo de individuo a buscar **"Cliente"** o **"Chofer".**

En _modo se especifica la función que va a desempeñar…

- **M -** para modificación, buscara tanto individuos habilitados como deshabilitados.
- **B -** busca solo individuos habilitados y gestionara la baja.
- **S -** solo devuelve los datos del individuo habilitado seleccionado al formulario que lo invoque.

Para este caso de uso. Se utilizan los siguientes StoredProcedures.

```
LJDG.buscar_chofer
LJDG.buscar_chofer_habilitado
LJDG.buscar_cliente
LJDG.buscar_cliente_habilitado
```

ABM Automóvil

Se utilizan dos formularios.

BuscarAuto.cs

```
publicBuscarAuto(char _modo)
```

Recibe el modo de trabajo...

- **M** – Para modificar, muestra todos los automóviles.
- **B** – Para Baja, solo muestra los automóviles habilitados.

AltaModiAuto.cs

```
publicAltaModiAuto(char _modo, int _idAuto)
```

Recibe el modo de trabajo...

- **A** – Para alta, ofrece todas las opciones de carga exigidas por el enunciado.
- **M** – Pre carga el automóvil pasado por **`_idAuto`**.

Para este caso de uso. Se utilizan los siguientes StoredProcedures.

```
LJDG.buscar_auto
LJDG.buscar_auto_habilitado
LJDG.baja_auto

LJDG.obtener_marcas
LJDG.obtener_descripcion_turnos
LJDG.alta_auto
LJDG.modi_auto
LJDG.obtener_auto
```

Registro de Viajes

Se utiliza el formulario.

RegistroViaje.cs

Se recolectan todos los datos necesarios, haciendo uso del formulario **BuscarIndividuo.cs.**
Se realizan todas las validaciones solicitadas por el enunciado, y al motor se le deja la responsabilidad de chequear los viajes duplicados del cliente con el SP **`LJDG.registrar_viaje`**

Para este caso de uso. Se utilizan los siguientes StoredProcedures.

```
LJDG.obtener_auto_habilitado_chofer
LJDG.validar_horarios_turno
LJDG.obtener_precio_viaje
LJDG.registrar_viaje
```

ABM Rol

Se utilizan tres formularios y las clasesRol.cs y Funcionalidad.cs.

AltaRol.cs

Se recolectan todos los datos necesarios del rol y las funcionalidades existentes para poder asociarlas al mismo, las mismas fueron obtenidas por el método **publicstaticList<Funcionalidad>obtenerFuncionalidades()**de la clase Funcionalidad.cs. Se realizan todas las validaciones solicitadas por el enunciado, y se procede a realizar el guardado del mismo mediante el método **publicstaticintinsertarRol(stringnombreRol)**de la clase Rol.cs y luego al método **publicstaticvoidinsertarFuncxRol(intidRol,intidFuncionalidad)**de la clase Funcionalidad.cs.

BajaRol.cs

Aquí se muestra un combo desplegable con todos los roles existentes que se encuentren habilitados.
Los mismos fueron obtenidos a través del método **publicstaticList<Rol>obtenerRoles()**de la clase Rol.cs. Una vez seleccionado un rol y dado de baja a través del formulario, se llamará al método**publicstaticvoideliminarRol(introlId)**perteneciente a la misma clase.

EditarRol.cs

Aquí se muestra un combo desplegable con todos los roles existentes (con el mismo método utilizado en el formulario BajaRol.cs). Al ser seleccionado uno, se habilitan los campos para ser editado y se permite eliminar o agregar nuevas funcionalidades asociadas que fueron obtenidas por el método **publicstaticList<Funcionalidad>obtenerFuncxRol(intidRol)**de la clase Funcionalidad.cs, así como habilitar un rol que se encontraba deshabilitado. Para salvar la información del rol editado, se llamará al método **publicstaticvoideditarRol(introlId, stringrolNombre, introlHabilitado)**de la clase Rol.cs.

Para este caso de uso. Se utilizan los siguientes StoredProcedures.

```
LJDG.obtener_roles
LJDG.crear_rol
LJDG.editar_rol
LJDG.eliminar_rol

LJDG.obtener_funcionalidades
LJDG.obtener_funcionalidadesxrol
LJDG.crear_funcxrol
```

ABM Turno

Se utilizan tres formularios y la clase Turno.cs.

AltaTurno.cs

Se recolectan todos los datos necesarios del turno. Se realizan todas las validaciones solicitadas por el enunciado, y se procede a realizar el guardado del mismo mediante el método **`publicstaticintinsertarTurno(stringdescripcion, decimal horaInicio, decimal horaFin, decimal valorKm, decimal precioBase)`**de la clase Turno.

BajaTurno.cs

Aquí se muestra un combo desplegable con todos los turnos existentes que se encuentren habilitados.
Los mismos fueron obtenidos a través del **`método publicstaticList<Turno>obtenerTurnos()`**de la clase Turno.cs. Una vez seleccionado un rol y dado de baja a través del formulario, se llamará al método**`publicstaticvoideliminarTurno(intturnoId)`**perteneciente a la misma clase.

EditarTurno.cs

Aquí se muestra un combo desplegable con todos los turnos existentes (con el mismo método utilizado en el formulario BajaTurno.cs). Al ser seleccionado uno, se habilitan los campos para ser editado y se permite editar o habilitar un turno que se encontraba deshabilitado. Para salvar la información del turno editado, se llamará al método **`publicstaticvoideditarTurno(intturnoId, stringdescripcion, decimal horaInicio, decimal horaFin, decimal valorKm, decimal precioBase, intturnoHabilitado)`**de la clase Turno.cs.

Para este caso de uso. Se utilizan los siguientes StoredProcedures.

```
LJDG.obtener_turnos
LJDG.crear_turno
LJDG.editar_turno
LJDG.eliminar_turno
```

Listado Estadístico

Estadistica.cs

A través del menú principal se accede a dicho formulario, en el tenemos un campo para ingresar el año de la consulta y un combo para seleccionar el trimestre. Ambos datos serán pasados a los StoredProcedures que nos devuelven el listado correspondiente a un datagrid de acuerdo lo que devuelva el **SELECT** de la query.
Con una estructura Switch se determina cual será ejecutado.

LJDG.choferes_mayor_recaudacion.sql
LJDG.choferes_viaje_mas_largo.sql
LJDG.cliente_auto.sql
LJDG.clientes_mayor_consumo.sql

Facturación

Facturacion.cs

En este formulario se deben ingresar los datos exigidos en el enunciado.
Fecha de inicio de y fin de fin, para determinar el rango de los viajes facturas. El buscador de individuos nos permite seleccionar al Cliente.
El sistema automáticamente mostrara en un datagrid todos los viajes realizados por ese cliente en ese periodo y adicionalmente el apellido del chofer, la patente del automóvil utilizado y el precio de dicho viaje. Al final se calcula el monto a facturar.
Para este objetivo se utiliza el StoredProcedure.

LJDG.viajes_cliente.sql

Al registrar la factura, se realizan las validaciones y si son correctas, se llama al método **publicstaticstringinsertarFactura(intfact_cliente, decimal fact_total, DateTimefact_fecha, DateTimefact_fecha_inicio, DateTimefact_fecha_fin)** de la clase Factura.cs que procederá a registrar la factura.
El StoredProcedure encargado del registro de la misma es

LJDG.crear_factura.sql

Rendición de cuenta del chofer

Rendicion.cs

En este formulario se deben ingresar los siguientes datos:
- Fecha de rendición, para determinar el rango de los viajes facturas a considerar en la rendición.
- El selector del turno que se quiere considerar en la rendición.
- El buscador de individuos nos permite seleccionar al Chofer.

El sistema muestra mediante un datagrid todos los viajes realizados por ese chofer en ese día y turno, apellido del cliente, la patente del automóvil utilizado y el precio de dicho viaje esto se realizaba mediante el stored procedure **`LJDG.viajes_chofer`**, el importe total se calcula dentro de la aplicación.

Por último la aplicación permite realizar la rendición mediante el llamado al stored procedure **`LJDG.crear_rendicion`** que al finalizar nos mostrará un mensaje informando el éxito (se agregó la rendición) o error de la operación (ya existe la rendición, no existen viajes en el día/turno elegido).

Registro de usuario

AltaUsuario.cs

En este formulario se deben ingresar los siguientes datos:
- Nombre de usuario.
- Contraseña y su repetición.
- El selector de rol

La selección del rol se obtiene de una lista completada con los roles cargados en la Base de Datos por medio de una consulta dentro de la aplicación.
La aplicación se encarga de realizar la verificación de la existencia del nombre de usuario por medio de una consulta a la Base de Datos.
Completada la verificación la aplicación realiza el alta del usuario por medio del stored procedure **`LJDG.alta_usuario`**.