**Gestión de Datos**

08-2030

**UberFRBA**

**Estrategia**



|  |  |  |
| --- | --- | --- |
| **Profesor** | REINOSA, Enrique |  |
| **Alumnos** | RODRIGUEZ, Luis |  |
|  | 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 3](#_Toc483893328)

[BDHandler.cs 3](#_Toc483893329)

[BDParametro.cs 3](#_Toc483893330)

[Usuario 4](#_Toc483893331)

[Usuario.cs 4](#_Toc483893332)

[Login 5](#_Toc483893333)

[Login.cs 5](#_Toc483893334)

[SeleccionRol.cs 5](#_Toc483893335)

[Búsqueda de Chofer / Cliente 6](#_Toc483893336)

[BuscarIndividuo.cs. 6](#_Toc483893337)

[ABM Automóvil 7](#_Toc483893338)

[BuscarAuto.cs 7](#_Toc483893339)

[AltaModiAuto.cs 7](#_Toc483893340)

[Registro de Viajes 8](#_Toc483893341)

[RegistroViaje.cs 8](#_Toc483893342)

# 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…

public DataTable **execSelectSP**(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.

public String **iniciarSesion**()

Utiliza el SP LJDG.iniciar\_sesion para validar los datos de inicio de sesión. La contraseña ya viaja encriptada con el algoritmo SHA256.

public List<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.

/\* Estos coméntenlo quien los hizo \*/

public static void darDeBaja(String userId)

public static void darDeAlta(String userId, String pass, int rol)

# 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](https://github.com/pycongroo/GDD2017-uber/blob/master/App/Logueo/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.

public **BuscarIndividuo**(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 Stored Procedures.

LJDG.buscar\_chofer

LJDG.buscar\_chofer\_habilitado

LJDG.buscar\_cliente

LJDG.buscar\_cliente\_habilitado

# ABM Automóvil

Se utilizan dos formularios.

## BuscarAuto.cs

public BuscarAuto(char \_modo)

Recibe el modo de trabajo…

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

## AltaModiAuto.cs

public AltaModiAuto(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 Stored Procedures.

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 Stored Procedures.

LJDG.obtener\_auto\_habilitado\_chofer

LJDG.validar\_horarios\_turno

LJDG.obtener\_precio\_viaje

LJDG.registrar\_viaje

# ABM Rol

Se utilizan tres formularios y las clases Rol.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 public static List<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 public static int insertarRol(string nombreRol) de la clase Rol.cs y luego al método public static void insertarFuncxRol(int idRol,int idFuncionalidad) 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 public static List<Rol> obtenerRoles()de la clase Rol.cs. Una vez seleccionado un rol y dado de baja a través del formulario, se llamará al método public static void eliminarRol(int rolId) 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 public static List<Funcionalidad> obtenerFuncxRol(int idRol)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 public static void editarRol(int rolId, string rolNombre, int rolHabilitado) de la clase Rol.cs.

Para este caso de uso. Se utilizan los siguientes Stored Procedures.

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 public static int insertarTurno(string descripcion, 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 public static List<Turno> obtenerTurnos()de la clase Turno.cs. Una vez seleccionado un rol y dado de baja a través del formulario, se llamará al método public static void eliminarTurno(int turnoId) 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 public static void editarTurno(int turnoId, string descripcion, decimal horaInicio, decimal horaFin, decimal valorKm, decimal precioBase, int turnoHabilitado) de la clase Turno.cs.

Para este caso de uso. Se utilizan los siguientes Stored Procedures.

LJDG.obtener\_turnos

LJDG.crear\_turno

LJDG.editar\_turno

LJDG.eliminar\_turno