

## ESPECIFICACION FUNCIONAL

## Guía Práctica - Semana 13

## PARTE 1: CONSULTAS PAGINADAS CON STORE PROCEDURE

## Competencia:

El desarrollo de esta parte de la guía práctica te permitirá la creación de consultas paginadas en entorno WEB en base a procedimientos almacenados.

## Descripción:

La presente especificación funcional se basa en la solución **DemoVentas\_3CapasWEB**, y se basa en una consulta de facturación en la base de datos VentasLeon con 3 criterios de filtrado en simultaneo: por cliente, por vendedor y por estado, cuyo resultado será en páginas de 50 registros, invocando a 2 store procedures para tal efecto:

- a) `usp_ListarFacturas_Paginacion`: Que maneja como parámetros los 3 criterios de filtro de facturas: cliente, vendedor y estado. Además, tiene un cuarto parámetro que es el número de página que se debe retornar (página de 50 registros) de las facturas que cumplan con las condiciones de los filtros.
- b) `usp_NumPag_ListarFacturas_Paginacion`: Que tiene 3 parámetros basados en las condiciones de filtro: cliente, vendedor y estado. Retorna en un parámetro de salida el número de registros que cumplen con las condiciones de filtro.

## Paso 1: Creando métodos en la capa de datos

Recupera tu solución **DemoVentas\_3CapasWEB**. En la capa de datos **ProyVentas\_ADO** dentro de la clase **FacturaADO** agregaremos los siguientes métodos:

```
public DataTable ListarFacturas_Paginacion
(String strCod_cli, String strCod_ven, String strEstado, int16 intNumPag)
{
    // Metodo que retorna una pagina de registros de acuerdo a los parametros
    // de cliente, vendedor y estado
    try
    {
        DataSet dts = new DataSet();
        cnx.ConnectionString = MiConexion.GetCnx();
        cmd.Connection = cnx;
        cmd.CommandType = CommandType.StoredProcedure;
        cmd.CommandText = "usp_ListarFacturas_Paginacion";
        cmd.Parameters.Clear();
        cmd.Parameters.AddWithValue("@Cod_cli", strCod_cli);
        cmd.Parameters.AddWithValue("@Cod_ven", strCod_ven);
        cmd.Parameters.AddWithValue("@Estado", strEstado);
        cmd.Parameters.AddWithValue("@NumPag", intNumPag);
        SqlDataAdapter miada = new SqlDataAdapter(cmd);
        miada.Fill(dts, "FacturasPaginacion");
        return dts.Tables["FacturasPaginacion"];
    }
    catch (SqlException ex)
    {
        throw new Exception(ex.Message);
    }
}
```

```
public Int16 NumPag_ListarFacturas_Paginacion (String strCod_cli, String strCod_ven, String strEstado)
{
    // Metodo que retorna la cantidad de registros devueltos de acuerdo a los parametros de cliente, vendedor y estado
    try
    {
        cnx.ConnectionString = MiConexion.GetCnx();
        cmd.Connection = cnx;
        cmd.CommandType = CommandType.StoredProcedure;
        cmd.CommandText = "usp_NumPag_ListarFacturas_Paginacion";
        cmd.Parameters.Clear();
        cmd.Parameters.AddWithValue("@Cod_cli", strCod_cli);
        cmd.Parameters.AddWithValue("@Cod_ven", strCod_ven);
        cmd.Parameters.AddWithValue("@Estado", strEstado);
        // Parametro de salida que devolvera la cantidad de registros de la consulta filtrada por
        // cliente, vendedor y estado
        cmd.Parameters.Add("@NumReg", SqlDbType.Int);
        cmd.Parameters["@NumReg"].Direction = ParameterDirection.Output;
        cnx.Open();
        cmd.ExecuteNonQuery();
        Int16 NumReg = Convert.ToInt16(cmd.Parameters["@NumReg"].Value);
        return NumReg;
    }
    catch (SqlException ex)
    {
        throw new Exception(ex.Message);
    }
    finally
    {
        if (cnx.State == ConnectionState.Open)
        {
            cnx.Close();
        }
    }
}
```

**Nota:** Si por alguna razón no tuvieras la clase FacturaADO, puedes agregarla y codificar el método indicado.

El primer método devuelve un objeto DataTable de a lo mucho 50 registros de facturación correspondientes al número de página solicitada, según los valores pasados como filtros.

El segundo método devuelve un valor entero con la cantidad de registros que cumplen con los valores pasados como filtros.

Dentro del mismo proyecto de capa de datos ProyVentas\_ADO, en la clase ClienteADO agregue el método siguiente:

```
public DataTable ListarCliente()
{
    try
    {
        DataSet dts = new DataSet();
        cnx.ConnectionString = MiConexion.GetCnx();
        cmd.Connection = cnx;
        cmd.CommandType = CommandType.StoredProcedure;
        cmd.CommandText = "usp_ListarCliente";
        cmd.Parameters.Clear();

        SqlDataAdapter ada = new SqlDataAdapter(cmd);
        ada.Fill(dts, "Clientes");

        return dts.Tables["Clientes"];
    }
    catch (SQLException ex)
    {
        throw new Exception(ex.Message);
    }
}
```

**Nota:** Si por alguna razón no tuvieras la clase ClienteADO, puedes agregarla y codificar el método indicado.

Este método retorna un objeto DataTable con el listado de clientes, lo cual nos permitirá llenar un Dropdown en la capa de presentación más adelante.

Por último, dentro del proyecto ProyVentas\_ADO y en la clase VendedorADO agregue el siguiente método:

```
public DataTable ListarVendedor()
{
    try
    {
        DataSet dts = new DataSet();
        cnx.ConnectionString = MiConexion.GetCnx();
        cmd.Connection = cnx;
        cmd.CommandType = CommandType.StoredProcedure;
        cmd.CommandText = "usp_ListarVendedor";
        cmd.Parameters.Clear();
        SqlDataAdapter ada = new SqlDataAdapter(cmd);
        ada.Fill(dts, "Vendedores");
        return dts.Tables["Vendedores"];
    }
    catch (SQLException ex)
    {
        throw new Exception(ex.Message);
    }
}
```

**Nota:** Si por alguna razón no tuvieras la clase VendedorADO, puedes agregarla y codificar el método indicado.

Este método retorna un objeto DataTable con el listado de vendedores, que al igual que el caso del método ListarCliente, nos permitirá llenar un Dropdown en la capa de presentación más adelante.

**Paso 2:** Codificando la capa de lógica de negocios

En el proyecto **ProyVentas\_BL**, dentro de la clase **FacturaBL** agregue los métodos siguientes, los cuales invocan a los que se implementaron en la clase **FacturaADO** en el paso anterior:

```
public DataTable ListarFacturas_Paginacion
(String strCod_cli, String strCod_ven, String strEstado, Int16 intNumPag)
{
    return objFacturaADO .ListarFacturas_Paginacion(strCod_cli,strCod_ven,strEstado,intNumPag);
}

public Int16 NumPag_ListarFacturas_Paginacion
(String strCod_cli, String strCod_ven, String strEstado)
{
    return objFacturaADO .NumPag_ListarFacturas_Paginacion (strCod_cli ,strCod_ven ,strEstado);
}
```

Nota: Si no tiene la clase **FacturaBL** la puede crear e inicie el código haciendo el using a **ProyVentas\_ADO** y a **System.Data**, para luego definir la instancia **objFacturaADO** de la clase **FacturaADO**, así:

```
// Agregar...
using ProyVentas_ADO;
using System.Data;
namespace ProyVentas_BL
{
    public class FacturaBL
    {
        FacturaADO objFacturaADO = new FacturaADO();
    }
}
```

En el mismo proyecto **ProyVentas\_BL**, dentro de la clase **ClienteBL** agregue el método siguiente, que invocara al que implemento en la clase **ClienteADO** en el paso anterior:

```
public DataTable ListarCliente()
{
    return objClienteADO.ListarCliente();
}
```

Nota: Si no tiene la clase **ClienteBL** la puede agregar e inicie el código haciendo el using a **ProyVentas\_ADO** y a **System.Data**, para luego definir la instancia **objClienteADO** de la clase **ClienteADO**, así:

```
using System.Threading.Tasks;
// Agregamos
using ProyVentas_ADO;
using System.Data;
namespace ProyVentas_BL
{
    public class ClienteBL
    {
        ClienteADO objClienteADO = new ClienteADO();
    }
}
```

Por último, en el mismo proyecto **ProyVentas\_BL**, dentro de la clase **VendedorBL** agregue el método siguiente, que invocará al que implemento en la clase **VendedorADO** en el paso anterior:

```
public DataTable ListarVendedor()
{
    return objVendedorADO.ListarVendedor();
}
```







**Nota:** Si no tiene la clase **VendedorBL** la puede agregar e inicie el código haciendo el using a **ProyVentas\_ADO** y a **System.Data**, para luego definir la instancia **objVendedorADO** de la clase **VendedorADO**, así:

```
// Agregar
using ProyVentas_ADO;
using System.Data;

namespace ProyVentas_BL
{
    public class VendedorBL
    {
        VendedorADO objVendedorADO = new VendedorADO();
    }
}
```

### Paso 3: Implementando la capa de presentación

Ubíquese en la capa de presentación, es decir, el proyecto **SitioWEB\_VentasGUI**. En la carpeta Consultas de dicho proyecto, copie y pegue los archivos del formulario WEB llamado **WebFacturacionPaginada**, que está dentro del material de la presente semana:

Nombre	Fecha de modificación	Tipo	Tamaño
 WebFacturacionCliente.aspx	24/11/2022 00:19	ASP.NET Server Pa...	10 KB
 WebFacturacionCliente.aspx.cs	24/11/2022 02:54	C# Source File	2 KB
 WebFacturacionCliente.aspx.designer.cs	23/11/2022 23:39	C# Source File	11 KB
 WebFacturacionPaginada.aspx	24/11/2022 02:18	ASP.NET Server Pa...	6 KB
 WebFacturacionPaginada.aspx.cs	24/11/2022 01:33	C# Source File	8 KB
 WebFacturacionPaginada.aspx.designer.cs	24/11/2022 00:21	C# Source File	4 KB

El formulario tiene el presente diseño:

ScriptManager - ScriptManager1

Principal (Personalizado)

**Listado paginado de Facturas**

Seleccione cliente: Sin enlazar  
 Seleccione vendedor: Sin enlazar  
 Seleccione estado: --Todos--

[bMensaje]

Filtrar

Nro. Factura	Fec. Facturacion	Fec. Cancelacion	Total(\$/.)	R.S. Cliente	RUC Cliente	Apellidos Vend.	Nombres Vend.
DataBound	DataBound	DataBound	DataBound	DataBound	DataBound	DataBound	DataBound
DataBound	DataBound	DataBound	DataBound	DataBound	DataBound	DataBound	DataBound
DataBound	DataBound	DataBound	DataBound	DataBound	DataBound	DataBound	DataBound
DataBound	DataBound	DataBound	DataBound	DataBound	DataBound	DataBound	DataBound
DataBound	DataBound	DataBound	DataBound	DataBound	DataBound	DataBound	DataBound

Sin er

Tiene 3 DropDownList para la selección de los filtros por cliente, vendedor y estado (este último ya está definido en tiempo de diseño con las opciones de los 3 estados Pendiente, Cancelada y Anulada). Tiene el botón etiquetado como Filtrar el ejecuta la consulta y muestra en el gridView la información solicitada. En la parte inferior izquierda hay un DropDownList para seleccionar que numero de página se desea visualizar.

Revise con detenimiento todo el código de servidor del ejemplo. Los comentarios incluidos lo ayudaran a entender el manejo del filtrado y paginación con store procedure. Antes de ejecutar no olvide quitar los comentarios lo referente al ProgressTemplate (encerrado en rojo en la siguiente figura) para que la ejecución incluya la visualización del gif de carga mientras se ejecuta el filtro:

```

</ContentTemplate>
<asp:UpdatePanel>
  <asp:UpdateProgress ID="UpdateProgress1" runat="server" DisplayAfter="0" AssociatedUpdatePanelID="UpdatePanel1">
    <%--ProgressTemplate>
      <div class="overlay">
        <div class="overlayContent">
          <h2>Cargando</h2>
          
        </div>
      </div>
    </ProgressTemplate-->%>
  </asp:UpdateProgress>
</asp:Content>

```

## PARTE 2: DESCARGA DE ARCHIVOS EXCEL CON EPPLUS

### Competencia:

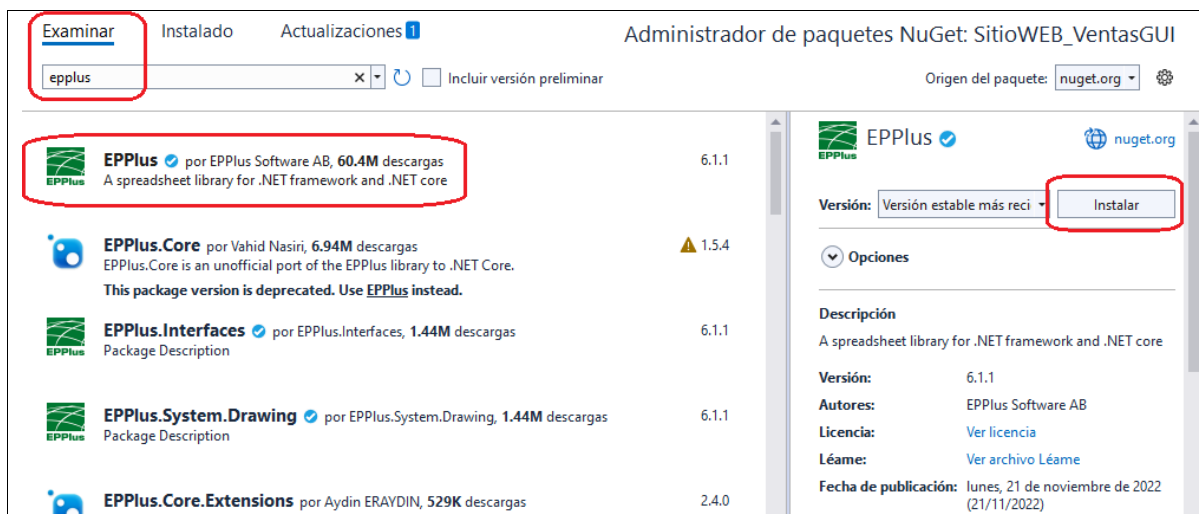
El desarrollo de esta parte de la guía práctica te permitirá generar archivos Excel a manera de reportes, tras efectuar una consulta

### Descripción:

Para este ejercicio debes haber implementado la consulta de facturación de clientes entre fechas mediante el formulario WebfacturacionCliente hecho en clase. El resultado de la consulta mostrada en el GridView ahora se podrá descargar en un archivo Excel.

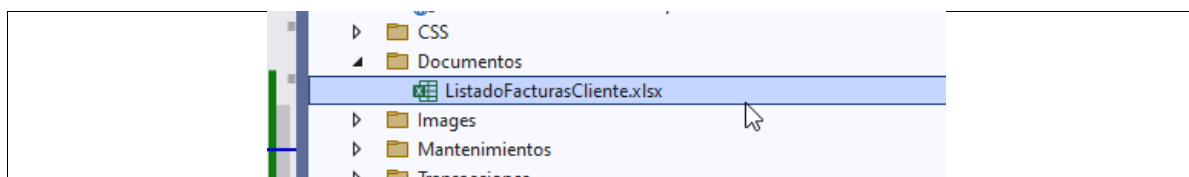
## Paso 1:

Instale Eppius en su proyecto WEB SitioWEB\_VentasGUI (tal como se hizo en el proyecto Windows anteriormente)



## Paso 2:

Dentro del sitio debe crear una carpeta llamada Documentos y copie dentro de esta el archivo ListadoFacturasCliente.xlsx proporcionado en su plantilla de la presente semana. Debería obtener esto:



## Paso 3:

Agregue el botón con el nombre “btnDescargar” en su formulario, tal como se indica a continuación:



## Paso 4:

Ahora vaya al formulario WebFacturacionCliente y en la ventana de código del servidor del mismo agregue las líneas enmarcadas dentro de los recuadros en rojo de la siguiente figura:

```

using ProyVentas_BL;
using ProyVentas_BE;
using System.Data;
// Para la descarga Excel (no olvide instalar EPPlus)
using System.IO;
using OfficeOpenXml;

namespace SitioWEB_VentasGUI.Consultas
{
    public partial class WebFacturacionCliente : System.Web.UI.Page
    {
        // Instancias ...
        ClienteBL objClienteBL = new ClienteBL();
        ClienteBE objClienteBE = new ClienteBE();
        FacturaBL objFacturaBL = new FacturaBL();

        protected void Page_Load(object sender, EventArgs e)
        {
            if (Page.IsPostBack == false)
            {
                // Agregamos estas 2 líneas para permitir la descarga del archivo Excel en el boton btnDescargar
                ScriptManager scriptManager = ScriptManager.GetCurrent(this.Page);
                scriptManager.RegisterPostBackControl(this.btnDescargar);

                // Definimos el contexto de la licencia del ExcelPackage como no comercial....
                ExcelPackage.LicenseContext = OfficeOpenXml.LicenseContext.NonCommercial;

                // El boton btnConsultar se inicia desactivado....
                btnConsultar.Enabled = false;
            }
        }
    }
}

```

## Paso 5:

En el evento click del botón **btnDescargar** codifique las siguientes líneas:

```

protected void btnDescargar_Click(object sender, EventArgs e)
{
    try
    {
        // Ruta del archivo plantilla del reporte en desarrollo
        String rutaarchivo = Server.MapPath("/") + @"Documentos\ListadofacturasCliente.xlsx";
        // Obtenemos las facturas
        DataTable dtFacturas = new DataTable();
        dtFacturas = objFacturaBL.ListarFacturasClienteFechas(txtCod.Text.Trim(),
            Convert.ToDateTime(txtFecIni.Text.Trim()), Convert.ToDateTime(txtFecFin.Text.Trim()));
        Int16 intRegistros = Convert.ToInt16 ( dtFacturas.Rows.Count);
        if (intRegistros == 0)
        {
            throw new Exception("No hay facturas registradas para el reporte. Verifique.");
        }
        // Fila de inicio del reporte
        Int16 fila1 = 5;
        using (var pck = new OfficeOpenXml.ExcelPackage(new FileInfo(rutaarchivo)))
        {
            //Nombre de archivo para descargar
            String filename = "ListadoFacturasCliente_" + DateTime.Today.ToShortDateString();
            ExcelWorksheet ws = pck.Workbook.Worksheets["Hoja1"];
            // Se imprime la cabecera
            ws.Cells[2, 4].Value = txtRazSoc.Text;
            ws.Cells[2, 6].Value = txtFecIni.Text + " y " + txtFecFin.Text;
            //Llenamos el Excel con las facturas
            foreach (DataRow drFactura in dtFacturas.Rows)
            {
                ws.Cells[fila1, 1].Value = drFactura["Num_fac"].ToString();

                DateTime fecfac = Convert.ToDateTime(drFactura["Fec_fac"]);
                ws.Cells[fila1, 2].Value = fecfac.ToShortDateString();
            }
        }
    }
}

```



```
DateTime feccan ;
if (drFactura["Fec_can"] == DBNull.Value)
{
    ws.Cells[fila1, 3].Value = "-";
}
else
{
    feccan = Convert.ToDateTime(drFactura["Fec_can"]);
    ws.Cells[fila1, 3].Value = feccan.ToShortDateString();
}

Single total = Convert.ToSingle(drFactura["Total"]);
ws.Cells[fila1, 4].Value = total.ToString("#,###,##0.00");
ws.Cells[fila1, 5].Value = drFactura["Ape_ven"].ToString() + "," + drFactura["Nom_ven"].ToString();
ws.Cells[fila1, 6].Value = drFactura["Estado"].ToString();
fila1 += 1;
}
ws.Cells[fila1, 1].Value = "Total registros : " + intRegistros.ToString();
//Modificando el ancho de las columnas
ws.Column(1).Width = 20;
ws.Column(2).Width = 30;
ws.Column(3).Width = 30;
ws.Column(4).Width = 30;
ws.Column(5).Width = 50;
ws.Column(6).Width = 25;

//Escribir de nuevo al cliente y descargar el archivo desde el navegador
Response.Clear();
Response.ContentType = "application/vnd.openxmlformats-officedocument.spreadsheetml.sheet";
Response.AddHeader("content-disposition", "attachment; filename=" + filename + ".xlsx");
using (var memoryStream = new MemoryStream())
{
    pck.SaveAs(memoryStream);
    memoryStream.WriteTo(Response.OutputStream);
}
Response.End();
}
}
catch (Exception ex)
{
    lblMensajePopup.Text = ex.Message;
    PopMensaje.Show();
}
}
```

Para probar, debe ingresar el código de un cliente (que existía lógicamente), ingresar las fechas de inicio y fin y hacer clic en el botón “Descargar”.

**Nota:** No olvide que, si publica esta aplicación en su IIS, deberá crear la carpeta Documentos dentro de la carpeta wwwroot, y asignarle opciones de control total, tal como se explicó en clases