

# Manual SWBFORMS

Índice	Páginas
1.- Introducción	3
2.- Configurando el ambiente de desarrollo	3
2.1.- Instalación y configuración de JDK	3
2.2.- Instalación y configuración de Tomcat	6
2.3.- Instalación y configuración de MongoDB	8
2.4.- Descargar SWBForms de GitHub	11
2.4.1 Descargando SWBForms mediante NetBeans	12
2.5.- Instalar SWBForms en NetBeans	16
2.5.1 Utilizar el proyecto preconstruido “SWBForms”	16
2.5.2 Añadir SWBForms a un proyecto Netbeans existente	19
3.- Formularios de datos	27
3.1.- fields	31
3.2.- links	33
3.3.- stype	36
3.4.- Tipos de datos	39
3.5.- Modificadores de datos	40
3.6.- SelectFields	41
3.7.- ShowIf	42
3.8.- Formularios anidados	42
3.9.- Ventanas de Edición	45
3.10.- onLoad	48
3.11.- DisplayFormat y FormatCellValue	48
4.- Organizar la apariencia de los formularios	52
4.1.- validator	52
4.2.- startRow	53
4.3.- width	55
4.4.- numCols	58
4.5.- titleOrientation	59
4.6.- mask	61
4.7.- hint	62
4.8.- showHintInField	63
5.- Añadir funcionalidad a los campos de un formulario	64
5.1.- Manejadores de eventos	64

6.- Tablas	67
6.1.- Filtros	67
6.2.- GridView	69
6.3.- Implementar catálogos	72
7.- Configuración del esquema de usuarios y permisos de acceso	73
7.1.- Usuario, Roles y Grupos de usuarios	75
7.2.- Gestionar usuarios y permisos	78
8.- DataProcessors	79
9.- DataServices	80
10.- Implementar bitácoras de seguridad	82

## **1.-Introducción**

En este manual se explica el desarrollo de aplicaciones Web mediante **SWBForms**, un conjunto de herramientas de desarrollo que facilita la creación de aplicaciones Web creada por **INFOTEC** (Centro de Investigación de Investigación e Innovación en Tecnologías de la Información y Comunicación).

**SWBForms** es la plataforma de la suite **SemanticWebBuilder** para desarrollo ágil, basada en Java y Javascript que permite construir aplicaciones de negocio responsivas en muy corto tiempo con gran fortaleza de seguridad y desempeño. Con **SWBForms** es posible crear en minutos listas tabulares con funcionalidades complejas como agregar, editar y eliminar registros en línea; paginación automática además de filtrado, ordenación, intercambio de columnas, etc; crear formularios completos, agregarles validaciones de manera muy sencilla y rápida.

Con **SWBForms** se puede desarrollar aplicaciones Web mediante Javascript y JSP. Para exemplificar el potencial para el desarrollo de aplicaciones de **SWBForms** en el transcurso de este manual se explica el desarrollo de una aplicación demo diseñada para el manejo de inventarios.

## **2.- Configurando el ambiente de desarrollo**

Para el uso de **SWBForms** se requiere de la instalación de los siguientes softwares de desarrollo requeridos para su uso: **Java JDK 8**, **Apache Tomcat 8** y **MongoDB**. A continuación, se describe la instalación de estos ambientes de desarrollo, paso previo para el uso de **SWBForms**.

**Nota importante:** Durante este manual se describe el desarrollo de aplicaciones Web en Windows y mediante el ambiente de desarrollo integrado (IDE) Netbeans.

### **2.1.- Instalación y configuración de JDK**

Para instalar la plataforma de desarrollo Java también conocida como JDK (Java Development Kit), se debe de descargar este software de la página de la compañía Oracle empresa encargada de desarrollar Java, para este tutorial se descargo Java de la dirección: <http://www.oracle.com/technetwork/es/java/javase/downloads/index.html> en su versión

8u161. Una vez descargado el archivo ejecutable (.exe) se ejecuta dicho archivo y se muestra una pantalla de bienvenida, ver figura 1.

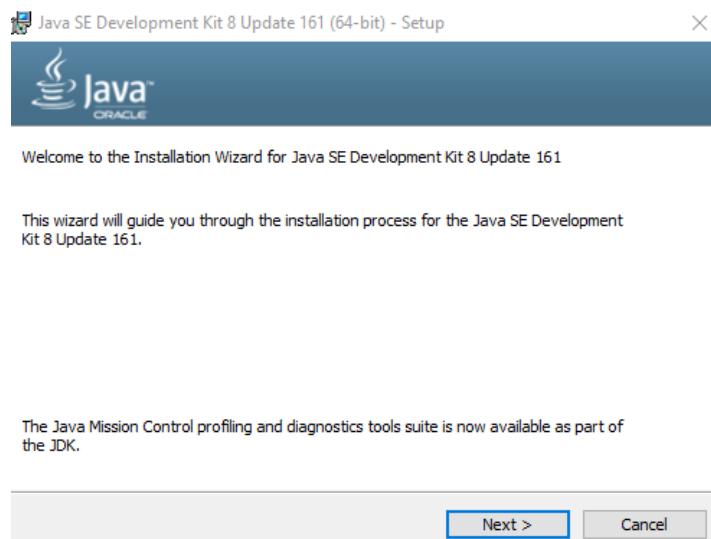


Figura 1. Pantalla de bienvenida del instalador de Java JDK.

Desde la pantalla que se muestra en la figura 1. se oprime el botón “Next” (Siguiente) y aceptando las configuraciones por defecto se obtiene finalmente una pantalla como se muestra en la figura 2, indicando que se a concluido satisfactoriamente la instalación de Java JDK.

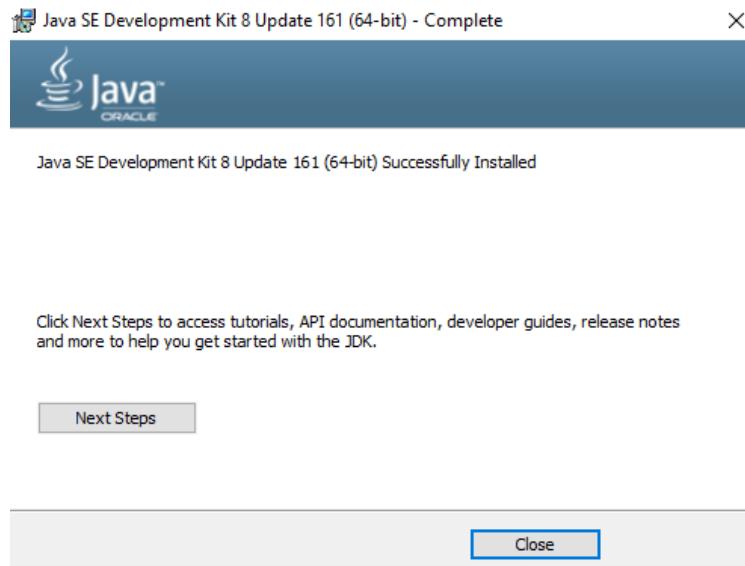


Figura 2. Pantalla de término exitoso del proceso de instalación de Java JDK.

Una vez instalado Java se debe de establecer las variables de entorno JAVA\_HOME y JRE\_HOME que son variables que usaran otros programas necesarios para el uso de **SWBForms** como Apache Tomcat. Para hacer la configuración de estas variables se debe de acceder a la ventana de configuración de variables de entorno de Windows accesible siguiendo la siguiente ruta: Panel de control -> Sistema y Seguridad -> Configuración avanzada de sistema -> Variables de ambiente (Environment Variables), ver figura 3.

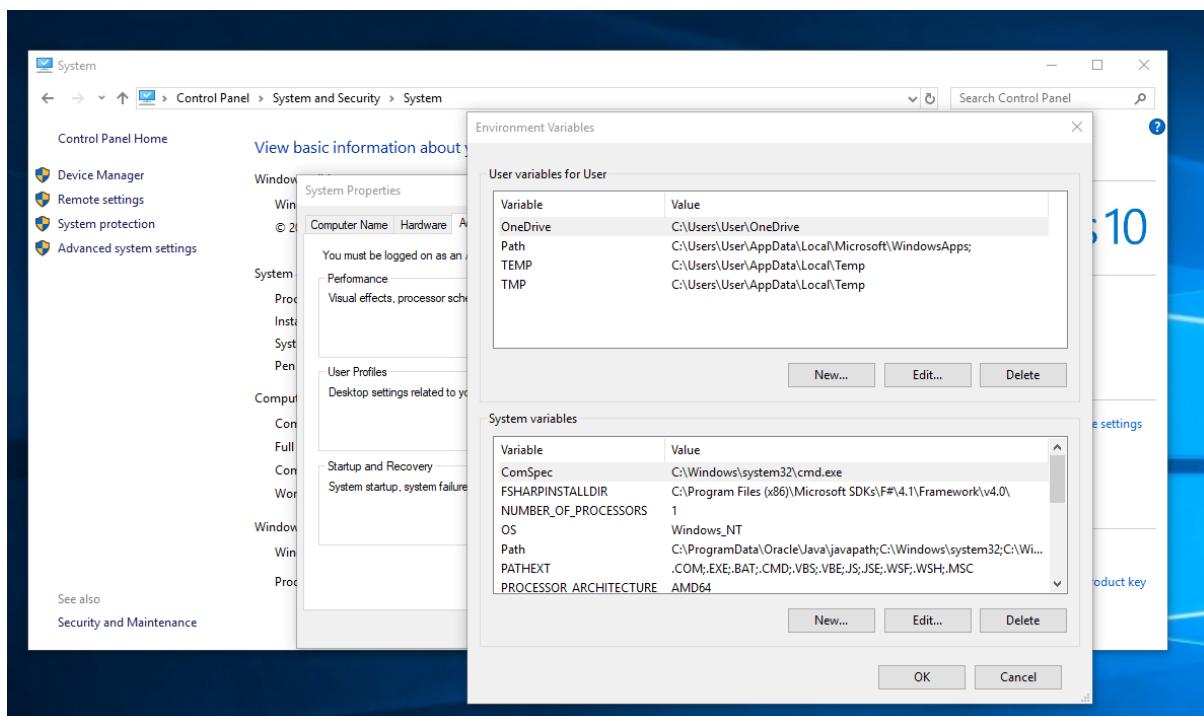


Figura 3. Pantalla de configuración de variables de ambiente en Windows 10.

Oprimiendo el botón “New” (Nuevo) dentro de la venta de configuración de variables de ambiente de Windows se puede agregar las nuevas variables y establecer sus valores en nuestro caso se agrega las siguientes dos variables: JAVA\_HOME y JRE\_HOME. Indicando como valor para cada variable la ruta donde quedo instalado Java, específicamente el JDK y JRE (Java Run-time Environment) que usualmente se instalan por defecto en la ruta archivo de programa -> Java , particularmente se debe de incluir como valor de estas variables algo similar a :

- **JAVA\_HOME**: C:\Program Files\Java\jdk<version>
  - **JRE\_HOME**: C:\Program Files\Java\jre<version>

En la figura 4 Se muestra como quedaron configuradas las variables de entorno en las rutas de defecto para la versión de Java instalada (en este ejemplo 8u161). Con esto se concluye el proceso de instalación y configuración de Java.

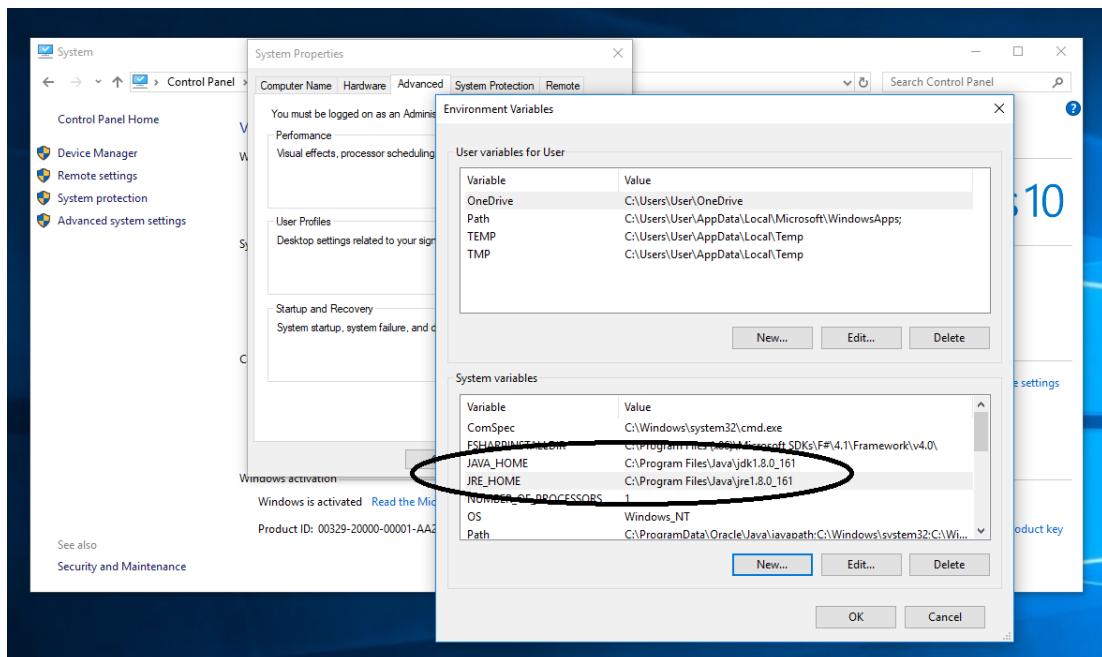


Figura 4. Variables de ambiente JAVA\_HOME y JRE\_HOME configuradas en Windows 10.

## 2.2.- Instalación y configuración de Apache Tomcat

Para el desarrollo de aplicaciones Web mediante **SWBForms** se requiere hacer uso de algún servidor de aplicaciones, en este manual se utilizará Apache Tomcat. Para instalar este servidor de aplicaciones primero se debe de bajar la versión comprimida (.ZIP) del sitio oficial de Apache Tomcat: <http://tomcat.apache.org/> . Es requerido instalar la versión 8 o 9 de Apache Tomcat, figura 5. Una vez que se a descargado se debe de descomprimir el archivo ZIP en alguna carpeta del sistema operativo de la máquina en la que se está instalando.

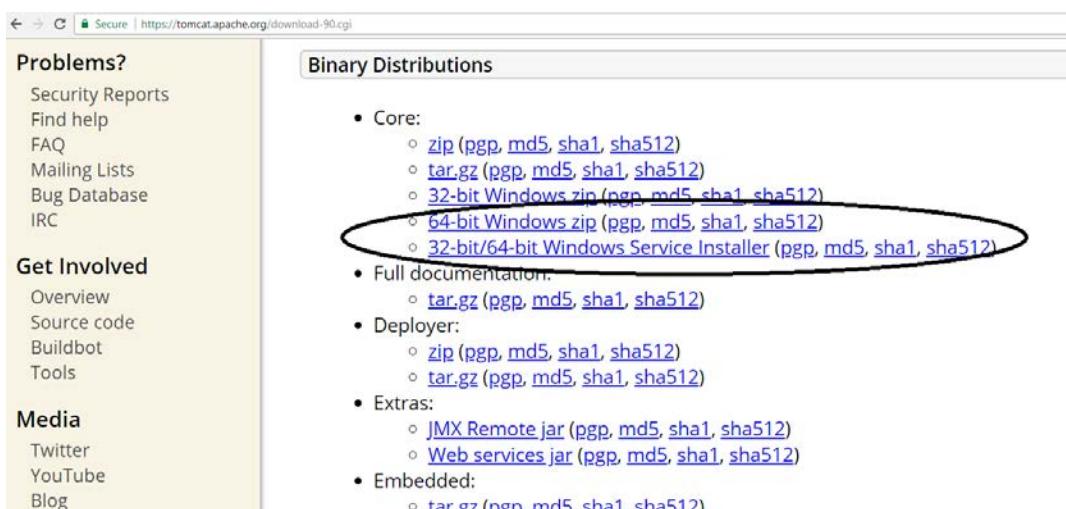


Figura 5. Versiones ZIP de Apache Tomcat que deben de ser consideradas para la instalación dependiendo si se esta usando la versión de Windows de 32 y 64 bits.

Posteriormente, se debe de establecer la variable de ambiente CATALINA\_HOME que contiene la ruta del directorio donde se descomprimió el archivo ZIP de Apache Tomcat. En la figura 6 se muestra la variable de ambiente ya configurada.

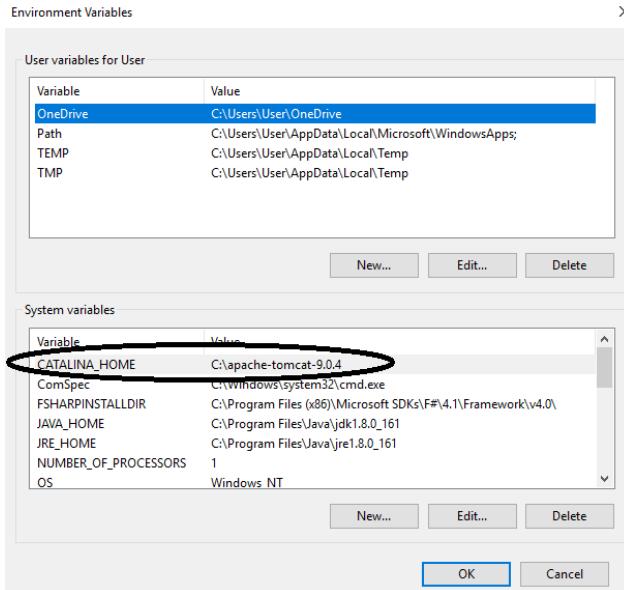


Figura 6. Variable CATALINA\_HOME ya configurada.

Finalmente, para comenzar el uso del servidor de aplicaciones y dejarlo listo para el desarrollo de aplicaciones con **SWBForms**, se debe de iniciar el servidor Apache Tomcat ejecutando en la línea de comandos el archivo: **startup.bat** que se encuentra en el directorio descomprimido de Apache Tomcat en el subdirectorio bin. Mostrando una pantalla como la que se muestra en la figura 7.

The screenshot shows a Windows Command Prompt window titled 'Administrator: Command Prompt'. The user runs the command 'C:\apache-tomcat-9.0.4\bin>startup.bat'. The output shows the Tomcat server starting up, with various log messages indicating the loading of the Catalina host configuration, deployment of web applications, and the start of the Coyote HTTP/1.1 protocol handler on port 8080. The window title is 'Tomcat'.

```

Administrator: Command Prompt
C:\apache-tomcat-9.0.4>
C:\apache-tomcat-9.0.4\bin>startup.bat
Using CATALINA_BASE: "C:\apache-tomcat-9.0.4"
Using CATALINA_HOME: "C:\apache-tomcat-9.0.4"
Using CATALINA_TMPDIR: "C:\apache-tomcat-9.0.4\temp"
Using JRE_HOME: "C:\Program Files\Java\jre1.8.0_161"
Using CLASSPATH: "C:\apache-tomcat-9.0.4\bin\bootstrap.jar;C:\apache-tomcat-9.0.4\bin\tomcat-juli.jar"
C:\apache-tomcat-9.0.4\bin>

[...]
Feb 07, 2018 10:27:40,149 INFO [main] org.apache.catalina.startup.Catalina.load Initialization processed in 10203 ms
Feb 07, 2018 10:27:40,038 INFO [main] org.apache.catalina.core.StandardService.startInternal Starting service [Catalina]
Feb 07, 2018 10:27:40,624 INFO [main] org.apache.catalina.core.StandardEngine.startInternal Starting Servlet Engine: Apache Tomcat/9.0.4
Feb 07, 2018 10:27:40,668 INFO [main] org.apache.catalina.startup.HostConfig.deployDirectory Deploying web application directory [C:\apache-tomcat-9.0.4\webapps\docs]
Feb 07, 2018 10:27:40,686 INFO [main] org.apache.catalina.startup.HostConfig.deployDirectory Deployment of web application directory [C:\apache-tomcat-9.0.4\webapps\host]
Feb 07, 2018 10:27:44,708 INFO [main] org.apache.catalina.startup.HostConfig.deployDirectory Deployment of web application directory [C:\apache-tomcat-9.0.4\webapps\examples]
Feb 07, 2018 10:27:50,983 INFO [main] org.apache.catalina.startup.HostConfig.deployDirectory Deployment of web application directory [C:\apache-tomcat-9.0.4\webapps\host-manager]
Feb 07, 2018 10:27:50,994 INFO [main] org.apache.catalina.startup.HostConfig.deployDirectory Deployment of web application directory [C:\apache-tomcat-9.0.4\webapps\host-manager]
Feb 07, 2018 10:27:51,224 INFO [main] org.apache.catalina.startup.HostConfig.deployDirectory Deployment of web application directory [C:\apache-tomcat-9.0.4\webapps\manager]
Feb 07, 2018 10:27:51,231 INFO [main] org.apache.catalina.startup.HostConfig.deployDirectory Deployment of web application directory [C:\apache-tomcat-9.0.4\webapps\manager]
Feb 07, 2018 10:27:51,797 INFO [main] org.apache.catalina.startup.HostConfig.deployDirectory Deployment of web application directory [C:\apache-tomcat-9.0.4\webapps\manager]
Feb 07, 2018 10:27:51,805 INFO [main] org.apache.catalina.startup.HostConfig.deployDirectory Deployment of web application directory [C:\apache-tomcat-9.0.4\webapps\manager]
Feb 07, 2018 10:27:51,955 INFO [main] org.apache.catalina.startup.HostConfig.deployDirectory Deployment of web application directory [C:\apache-tomcat-9.0.4\webapps\ROOT]
Feb 07, 2018 10:27:51,973 INFO [main] org.apache.coyote.AbstractProtocol.start Starting ProtocolHandler ["http-nio-8080"]
Feb 07, 2018 10:27:52,033 INFO [main] org.apache.coyote.AbstractProtocol.start Starting ProtocolHandler ["ajp-nio-8009"]
Feb 07, 2018 10:27:52,036 INFO [main] org.apache.catalina.startup.Catalina.start Server startup in 11837 ms

```

Figura 7. Servidor Apache Tomcat ya inicializado mediante el comando: startup.bat

Finalmente podemos verificar el funcionamiento del servidor de aplicaciones Apache Tomcat accediendo mediante un navegador a la página: <http://localhost:8080/> obteniéndose como respuesta una pantalla como la que se observa en al figura 8.

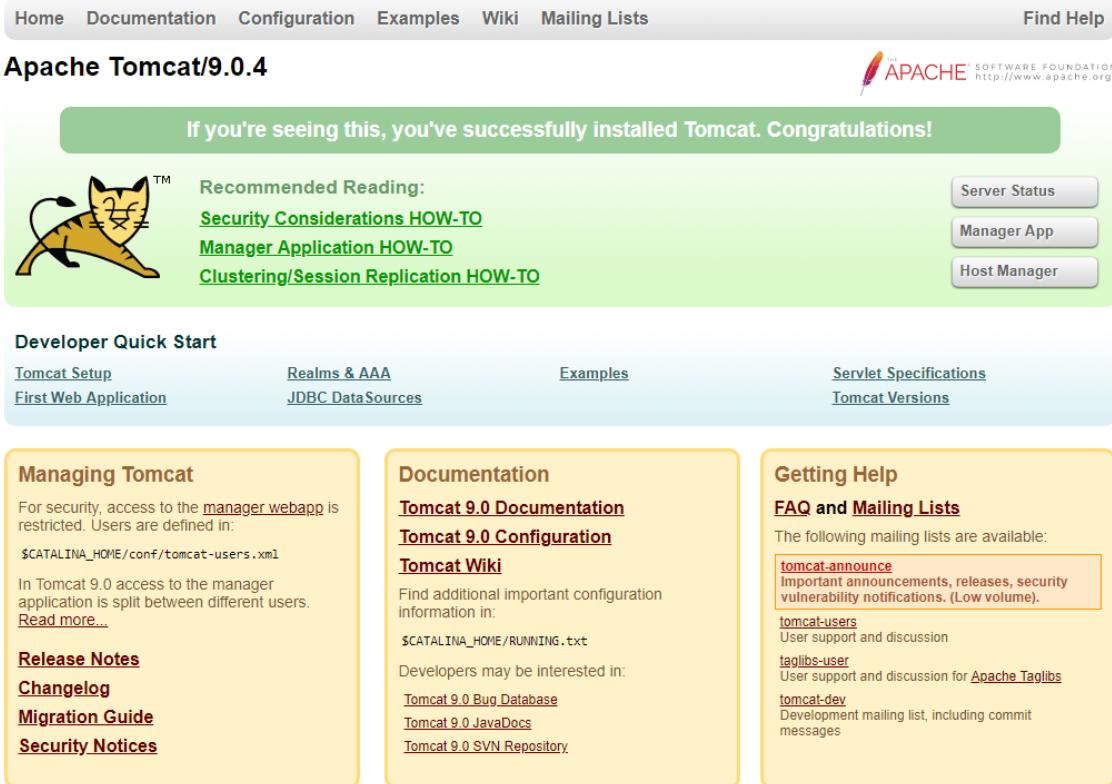


Figura 8. Página Web inicial de Apache Tomcat recién configurado.

### 2.3.- Instalación y configuración de MongoDB

Como base de datos para el desarrollo de aplicaciones Web mediante SWBForms se utilizará en este manual MongoDB aunque SWBForms puede ser usado con otras bases d datos. Para instalar MongoDB para su uso con SWBForm se debe de bajar el archivo ZIP de la versión de MongoDB más reciente en este caso a la fecha de realización del manual se instaló la versión 3.6.2 de 64 bits para Windows. La última versión de mongoDB puede ser bajada de la página: <https://www.mongodb.com/download-center#enterprise> , ver figura 9. Una vez que se ha descargado la carpeta comprimida de la versión de MongoDB (version archive) como se muestra en la figura 10 que se desea instalar se descomprime en una directorio en la maquina donde se haya instalado el servidor de aplicaciones Apache Tomcat. Previo a inicializar la operación de la base de datos MongoDB se requiere crear el directorio **db** que es el directorio por defecto en el cual se almacenan los datos en MongoDB para crear el directorio se puede usar el siguiente comando: md \data\db . Finalmente, para inicializar la operación de la base de datos MongoDB se ejecuta desde la línea de comandos el ejecutable mongod.exe que se

encuentra dentro de la carpeta descomprimida de mongoDB en la subcarpeta bin, como se observa en la figura 11.

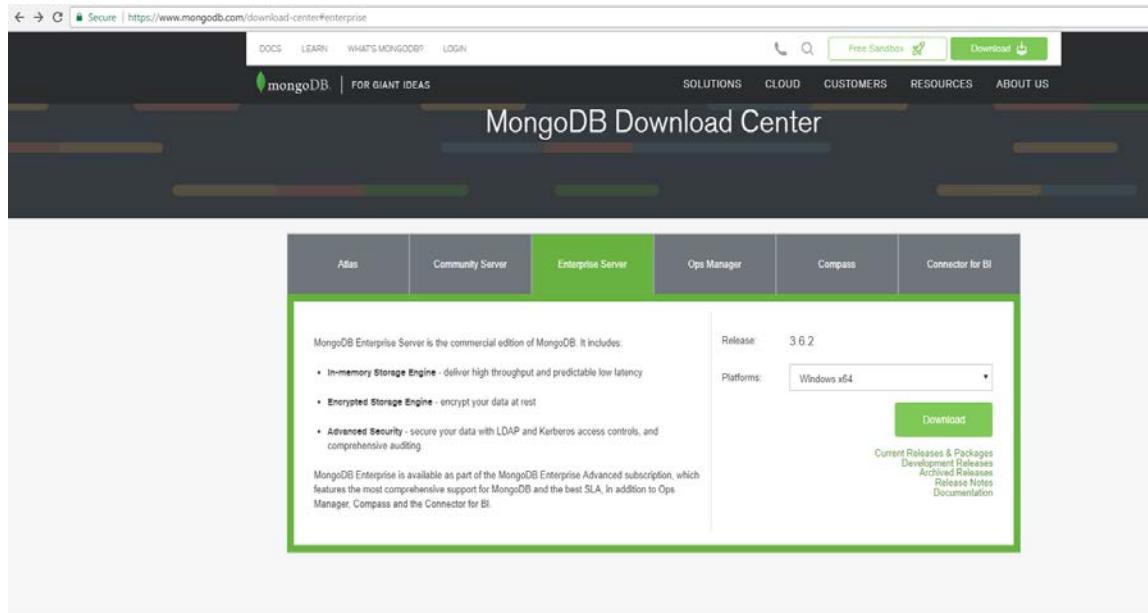


Figura 9. Página Web que permite la descarga de MongoDB en sus distintas versiones.

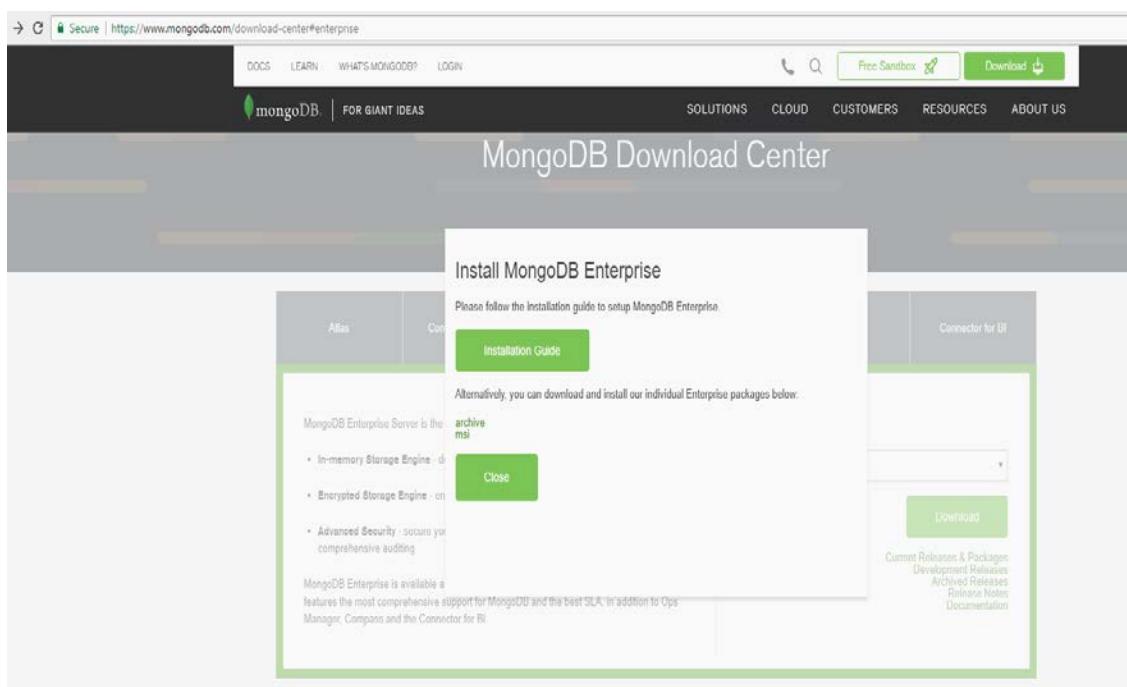


Figura 10. Se selecciona la versión archive de MongoDB para su instalación y uso de SWBForms.

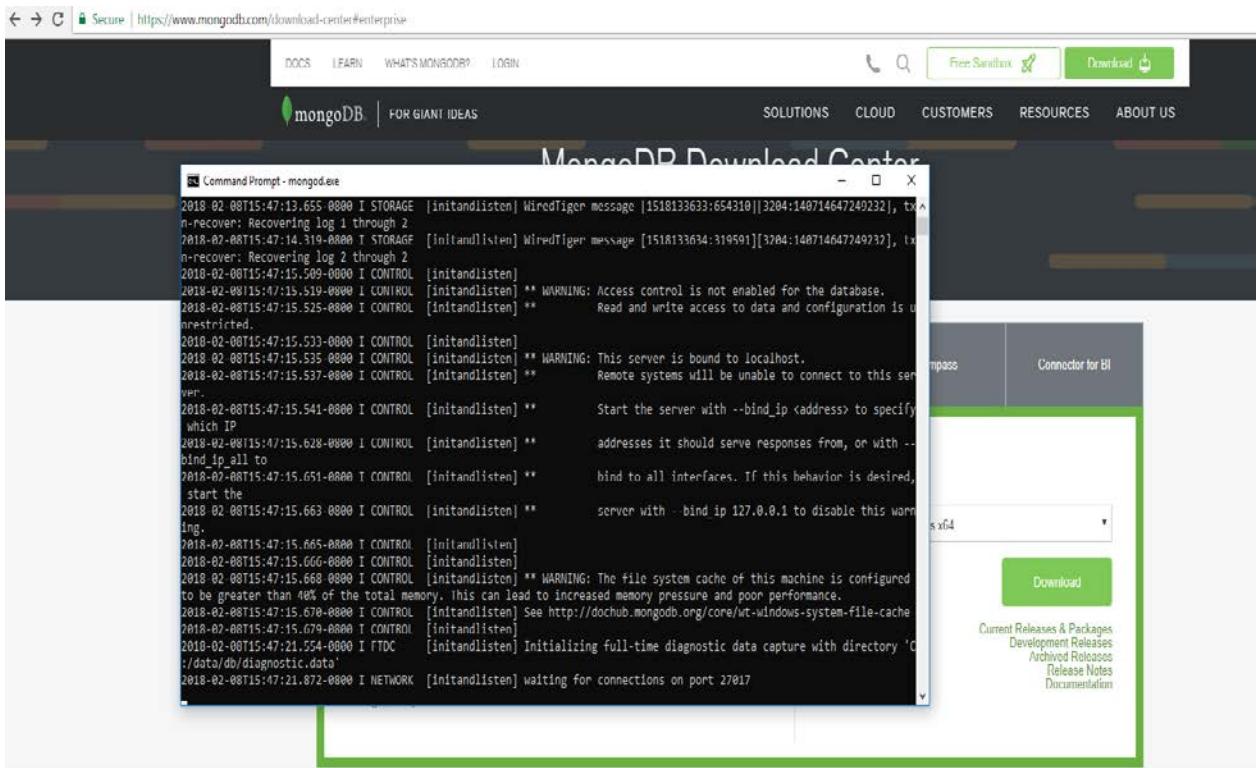


Figura 11. Se selecciona la versión archive de MongoDB para su instalación y uso de SWBForms.

## 2.4.- Descargar SWBForms desde GitHub

**SWBForms** es accesible mediante la herramienta de desarrollo colaborativo de software GitHub (<https://github.com>) en la dirección específica <https://github.com/SWBForms>, una vez que se ha ingresado a la página GitHub de **SWBForms** se puede observar una pantalla como la que se muestra en la figura 12.

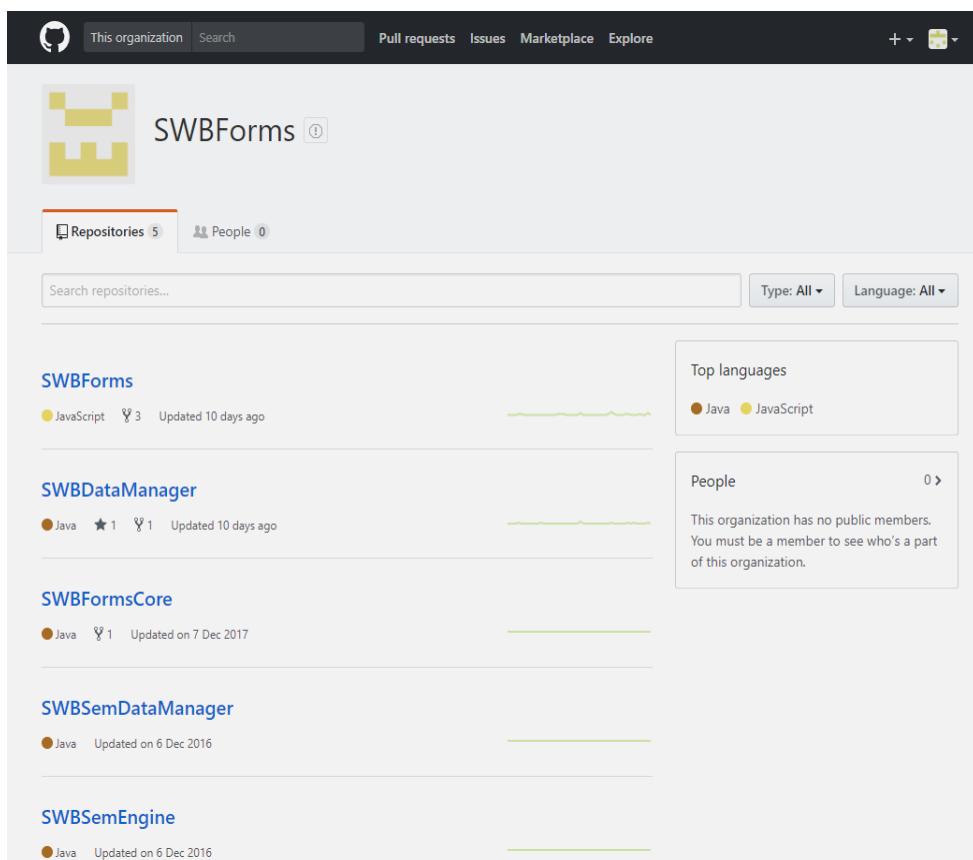


Figura 12. Pantalla en GitHub de **SWBForms**

En la página de github <https://github.com/SWBForms> que se muestra en la figura 12, se encuentran el conjunto de herramientas SWBForms y desde esta página pueden ser descargadas para su uso estas herramientas mediante algún IDE como Netbeans. En el caso de Netbeans este proceso se puede realizar de la siguiente forma:

#### 2.4.1.- Descargando SWBForms mediante Netbeans

Dentro de la pantalla principal de Netbeans se selecciona el menú “Team” (Equipo) y dentro de este el submenú “Git” y dentro de esta la opción “Clone” (Clonar). Como se muestra en la Figura 13.

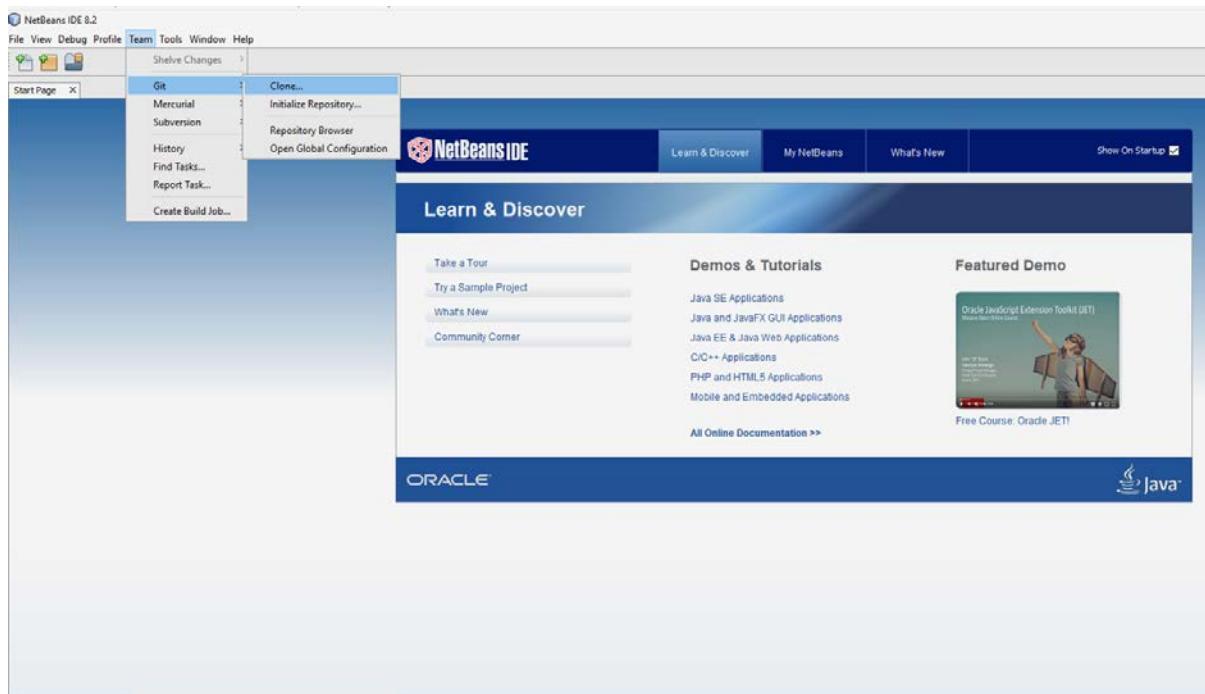


Figura 13. Seleccionando la opción “Clone (Clonar)” dentro del menú principal de Netbeans

Una vez seleccionada la opción “clone (Clonar)” se muestra una pantalla como en la Figura 14, donde se indica la dirección del repositorio GitHub del proyecto que se desea importar a Netbeans. Por ejemplo, si se desea descargar la aplicación Web maestra para el desarrollo de proyectos conocida como SWBForms del repositorio principal GitHub de **SWBForms**, ver figura 15, se ingresa la dirección .git del proyecto a descargar. Esta dirección web que se obtiene al ingresar a la liga del proyecto dentro del repositorio GitHub en este caso la dirección correspondiente es: <https://github.com/SWBForms/SWBForms.git> , ver figura 15.

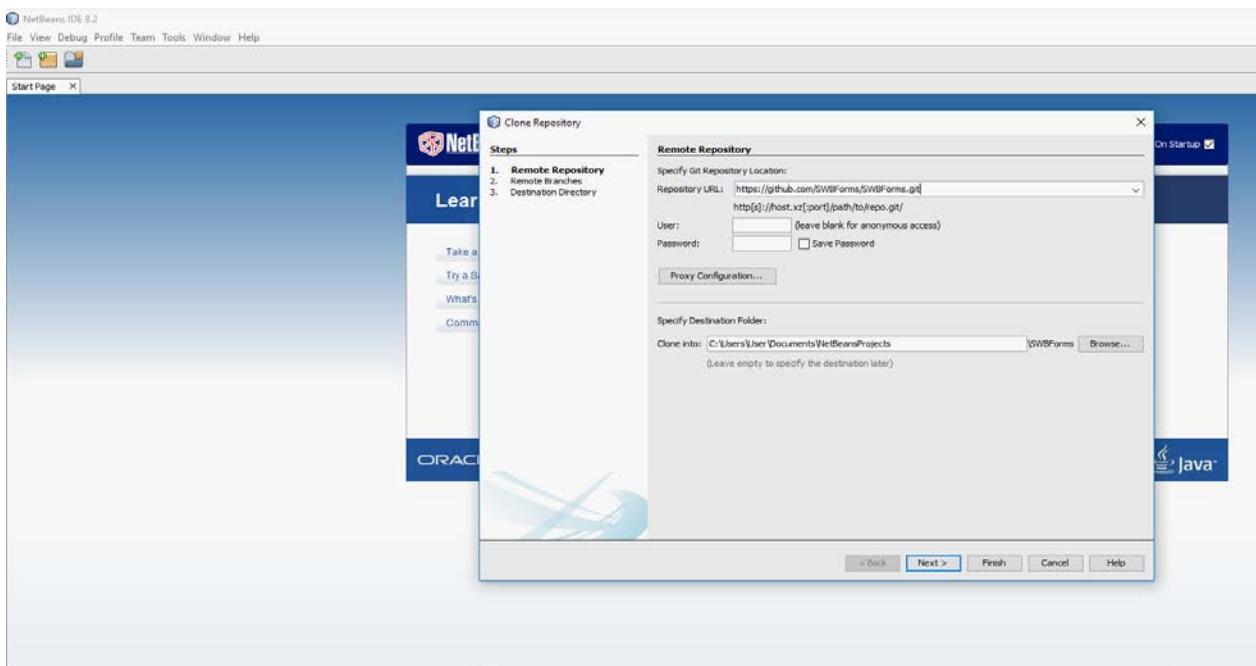


Figura 14. Indicando la dirección del software a descargar desde Github y el lugar donde se

No description, website, or topics provided.

28 commits	2 branches	0 releases	1 contributor
Branch: master	New pull request	Find file	Clone or download
<b>javersolis Isomorphic new Version</b> src/main Isomorphic new Version .gitignore first commit README.md first commit nb-configuration.xml first commit pom.xml Dependence resolve			
<b>Clone with HTTPS</b> Use Git or checkout with SVN using the web URL. <a href="https://github.com/SWBForms/SWBForms.git">https://github.com/SWBForms/SWBForms.git</a>			
<a href="#">Open in Desktop</a>		<a href="#">Download ZIP</a>	

almacenará a nivel local.

Figura 15. Pantalla de la página GitHub del proyecto SWBForms a descargar y el cuadro que contiene la dirección .git para su descarga que se obtiene al oprimir el botón “Clone or download”.

Una vez que se ha ingresado la URL del proyecto que se desea importa se especifica si es el caso el directorio donde se guardará el código descargado. En la figura 14 se deja el directorio por defecto sugerido por NetBeans.

Posteriormente, se muestra una pantalla donde se solicita seleccionar las versiones que se desean instalar de la aplicación que se está descargando, para este ejemplo se dejan las opciones por defecto, ver figura 16.

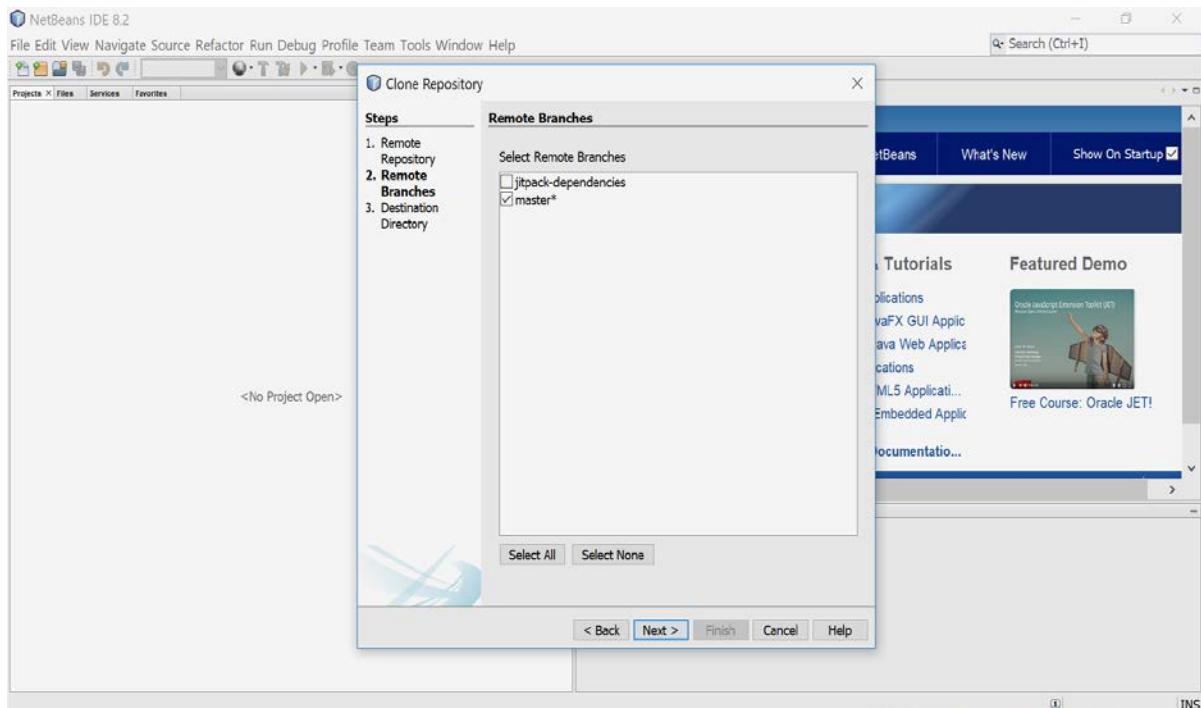


Figura 16. Pantalla de selección de versiones a descargar de una aplicación desde GitHub.

Una vez que se seleccionó la opción “siguiente” se muestra una pantalla donde nuevamente se pide confirmar el nombre de la aplicación y el directorio en el cual será instalada. Dejando las opciones por defecto se comienza con el proceso de descarga de la aplicación y su instalación para su uso mediante NetBeans, ver figura 17.

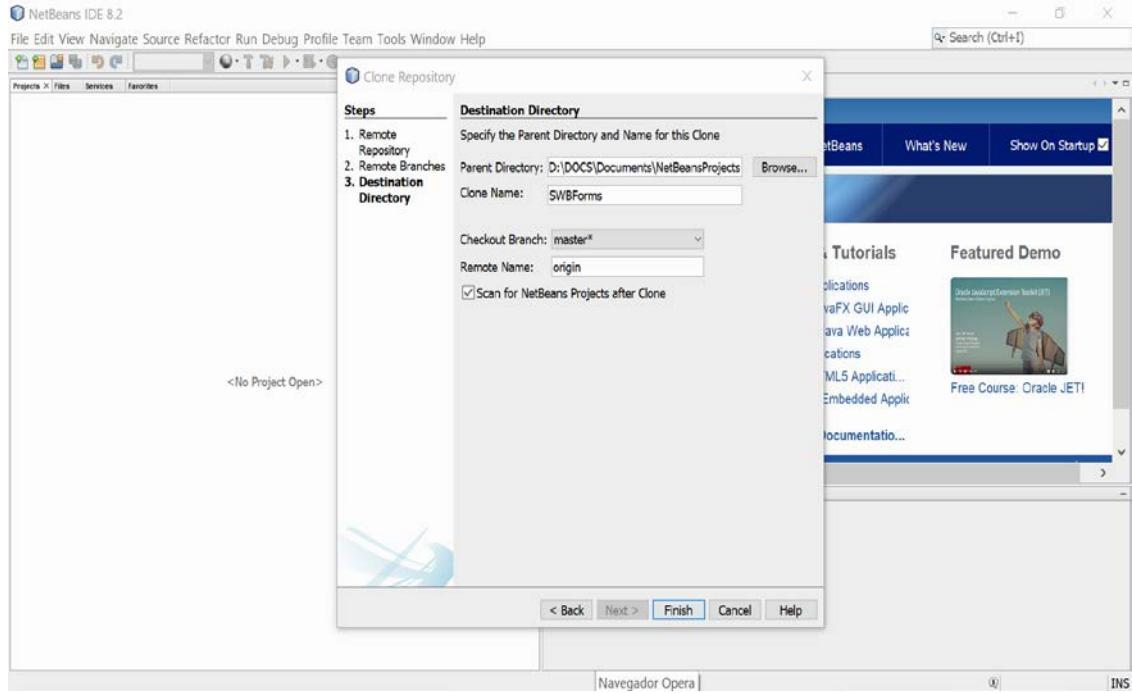


Figura 17. Conformación del directorio de instalación del proyecto descargado desde GitHub.

Finalmente, veremos la estructura del proyecto que se ha importado dentro de Netbeans desde GitHub. En este ejemplo se descargó de GitHub el proyecto SWBForms que es un esqueleto para el desarrollo de aplicaciones Web mediante **SWBForms**, ver figura 18 y 19.

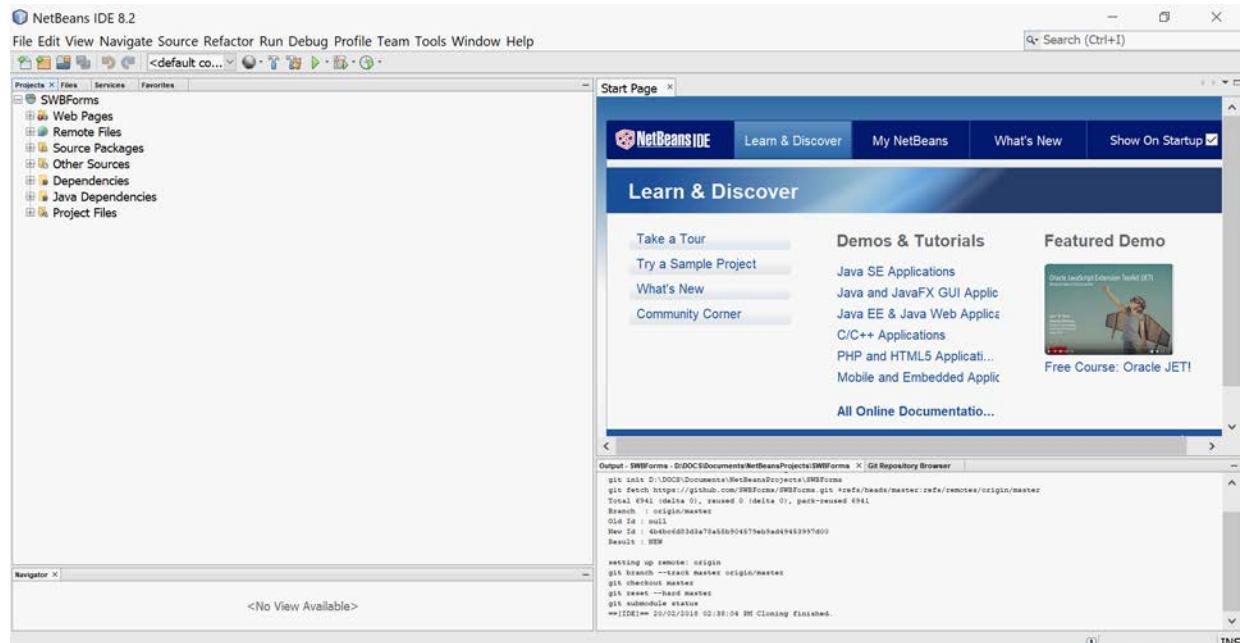


Figura 18. Proyecto SWBForms ya importado en Netbeans.

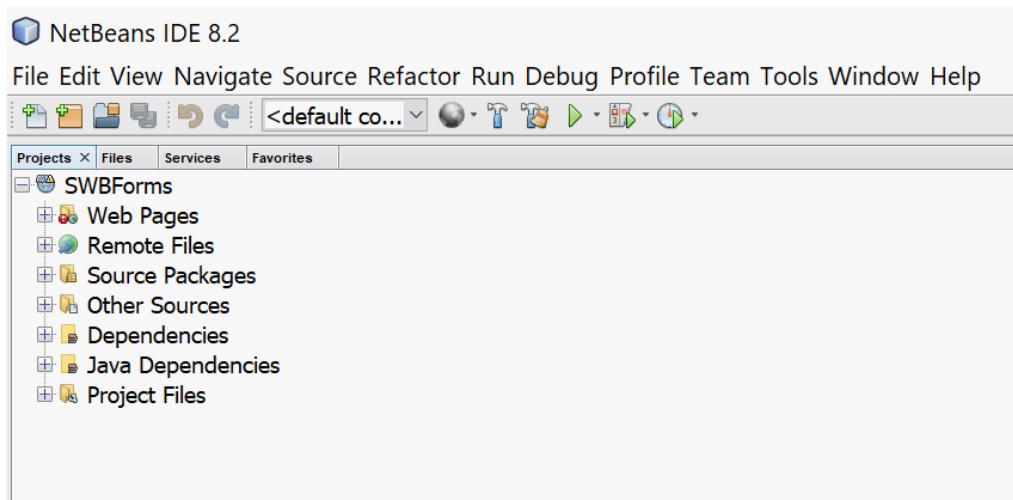


Figura 19. Detalle de la estructura del proyecto SWBForms ya importada en Netbeans.

Con esto se concluye el proceso de descarga e instalación de una aplicación mediante GitHub.

## 2.5.- Instalar SWBForms

Es posible utilizar las herramientas de desarrollo Web de **SWBForms** de dos formas distintas:

1. Utilizar el proyecto preconstruido “SWBForms”
2. Añadir SWBForms a un proyecto Netbeans existente.

### 2.5.1 Utilizar el proyecto preconstruido “SWBForms”.

La forma más sencilla de desarrollar aplicaciones Web mediante SWBForms es descargando la aplicación modelo que se provee en el página de GitHub de SWBForms (<https://github.com/SWBForms>). Esta aplicación maestra con nombre SWBForms contiene toda la estructura necesaria para el desarrollo de aplicaciones Web. Para descargar el proyecto SWBForms de la página GitHub se siguen los pasos del inciso 2.4. Posteriormente se puede ejecutar la aplicación para ver la pantalla de inicio de esta aplicación modelo de SWBForms, ver figura 20.

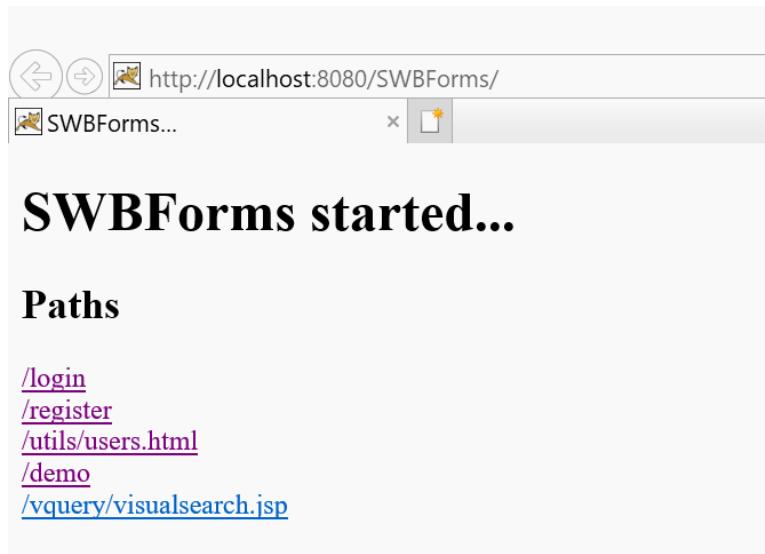


Figura 20. Pantalla de inicio de la aplicación modelo de SWBForms.

Cabe observar que para ejecutar la aplicación modelo para el desarrollo de aplicaciones SWBForms se debe tener activos tanto el servidor de aplicaciones TomCat y MongoDB.

Esta aplicación modelo una vez instalada puede ser modificada y adaptada para el desarrollo de cualquier aplicación Web que haga un uso extensivo de formularios y catálogos, como se muestra en secciones posteriores de este manual. Por ejemplo, si se desea se puede cambiar el nombre de la aplicación para comenzar a personalizar el proyecto se selecciona el nombre SWBForms de la estructura mostrada en la figura 19 y oprimiendo el botón izquierdo se selecciona la opción renombrar (“Rename”), como se muestra en la figura 21.

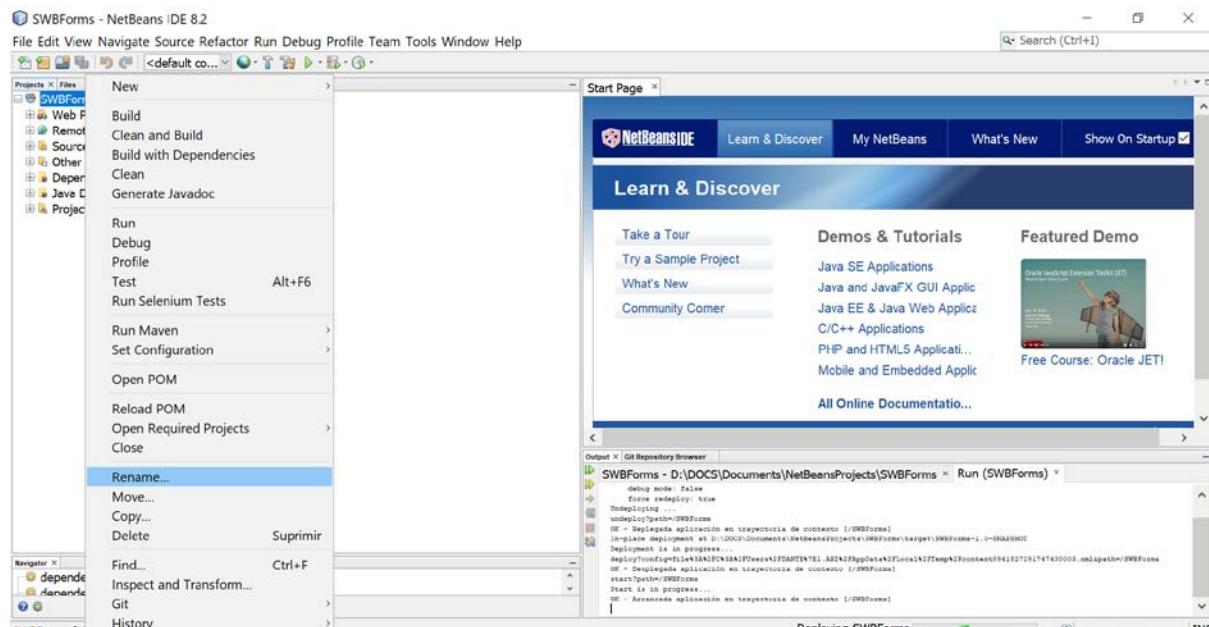


Figura 21. Seleccionando la opción “renombrar” para cambiar el nombre a la aplicación modelo de SWBForms.

Una vez seleccionada la opción renombrar se muestra una pantalla como en la figura 22. En esta pantalla se puede cambiar no solo el nombre de la aplicación sino también el nombre del artefacto y del folder. En este ejemplo se cambian todos los parámetros a Demo que es el nombre de la aplicación que desarrollara durante este manual para mostrar las capacidades de desarrollo de aplicaciones de SWBForms.

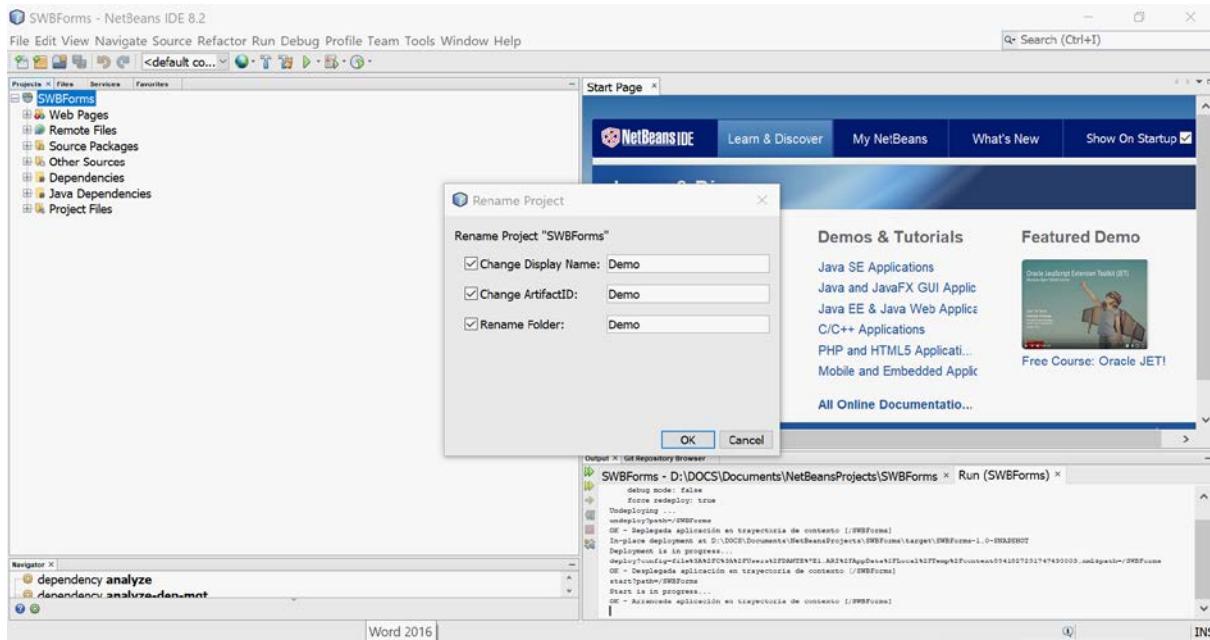


Figura 22. Opciones de cambio dentro de la opción renombrar en NetBeans.

Finalmente podemos observar como se ha cambiado el nombre del proyecto de la aplicación modelo de SWBForms a Demo, ver figura 23.

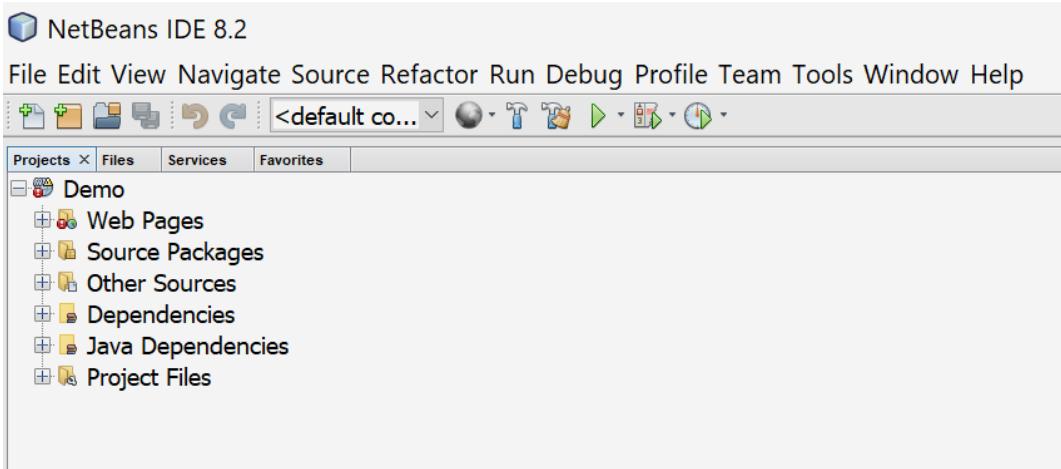


Figura 23. Estructura de la aplicación modelo de SWBForms con el nombre cambiado a Demo.

## 2.5.2 Añadir SWBForms a un proyecto Netbeans existente.

Para comenzar con el desarrollo de una aplicación Web con **SWBForms** desde un proyecto Netbeans existente se requiere realizar los siguientes pasos:

### Paso 2.5.2.1: Crear una nueva aplicación Web Maven mediante Netbeans.

Para iniciar el desarrollo de una aplicación Web mediante **SWBForms** se requiere crear una nueva aplicación Web Maven en Netbeans. Para realizar esto dentro de la pantalla principal de Netbeans, ver figura 15, se selecciona del menú File (Archivo) la opción New Project (Nuevo Proyecto). Una vez que se selecciona un nuevo proyecto aparece una pantalla en donde se debe de seleccionar dentro de la opción Categories (Categorías) que se desarrollará una aplicación Maven y dentro de la opción Projects (Proyectos) que será de tipo Web (Web Application (Aplicación Web)), como se observa en la figura 24.

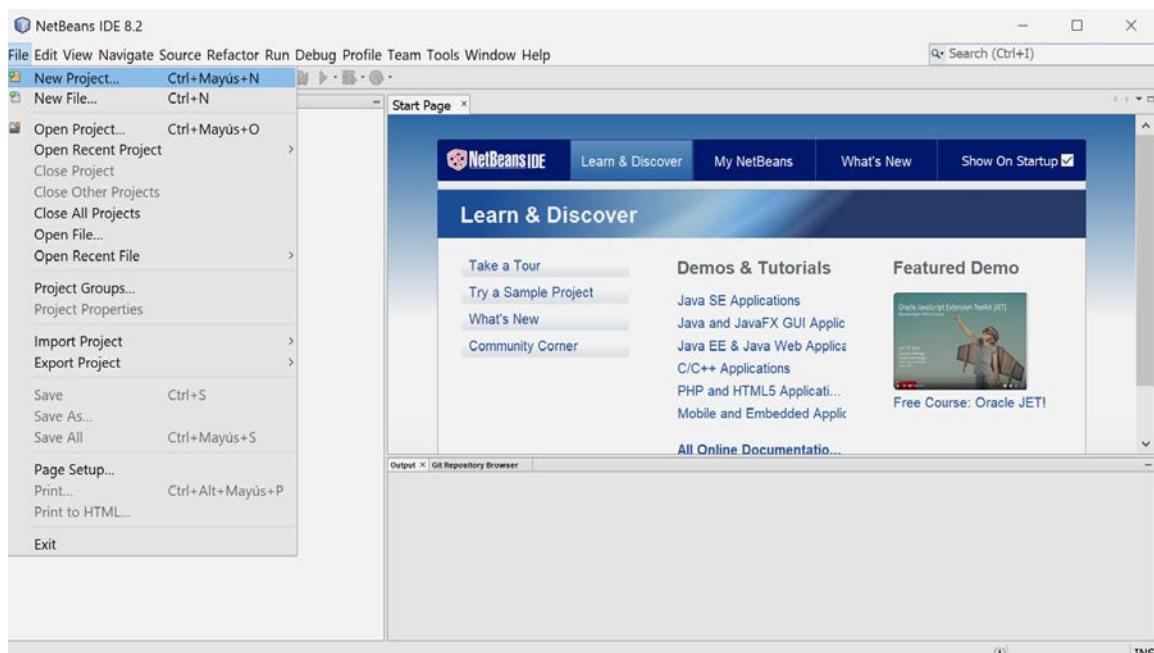


Figura 24. Creando un nuevo proyecto en Netbeans.

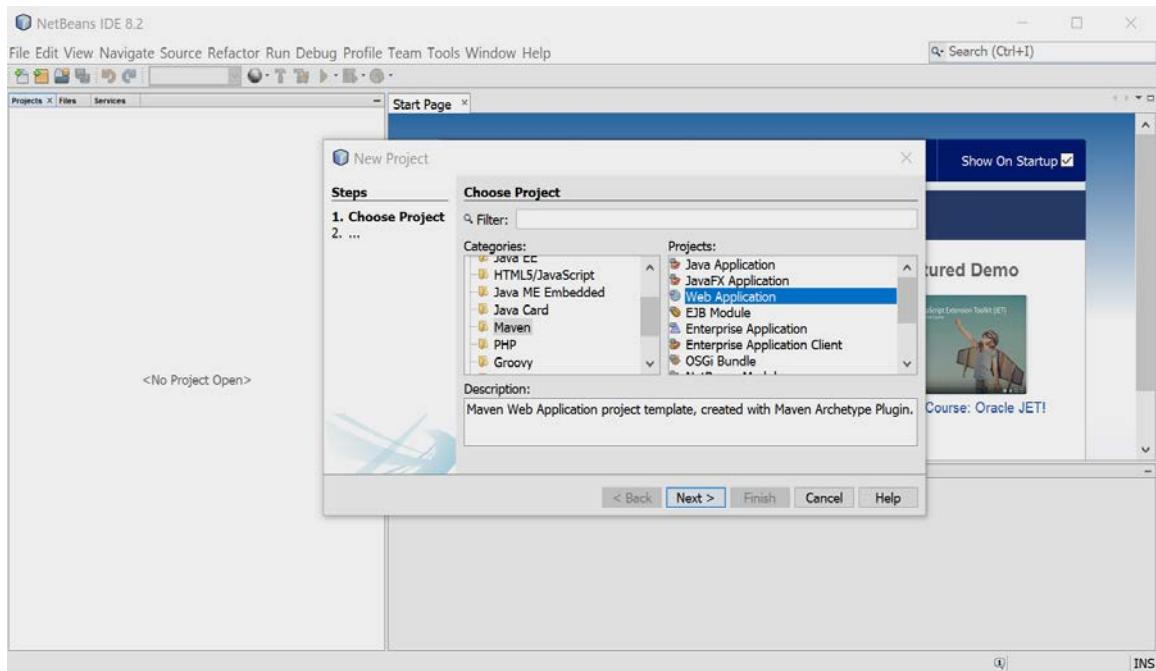


Figura 25. Creando una nueva aplicación Web Maven en Netbeans.

Una vez que se selecciona la creación de una aplicación Web dentro la pantalla Nuevo proyecto se oprime el botón Next (Siguiente), ver figura 25. Como resultado de esto se muestra una pantalla de configuración en donde se define el nombre de la aplicación y el directorio de trabajo, ver figura 26. En este ejemplo se define el nombre de la aplicación como “Demo” y se dejan las opciones por defecto.

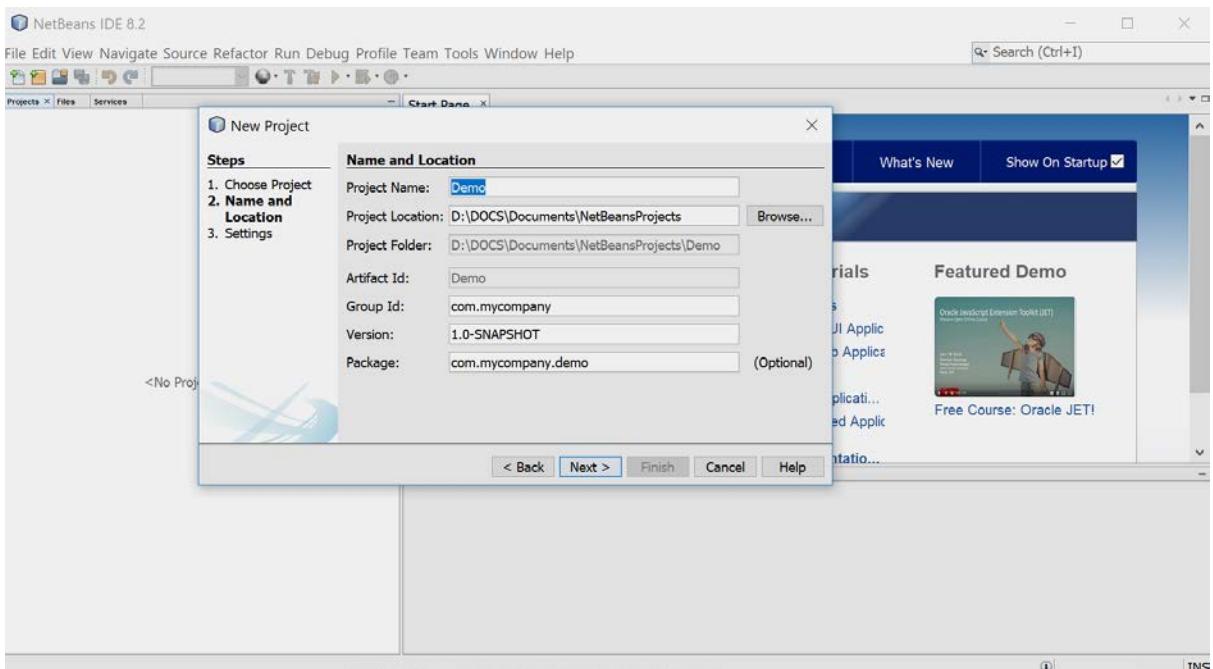


Figura 26. Asignando un nombre a una aplicación Web.

Oprimiendo de nueva cuenta el botón Next (Siguiente) mostrándose dos opciones más de configuración que son la definición del servidor de aplicaciones que usará (se selecciona Apache Tomcat) y de la versión de Java a utilizar, ver figura 27.

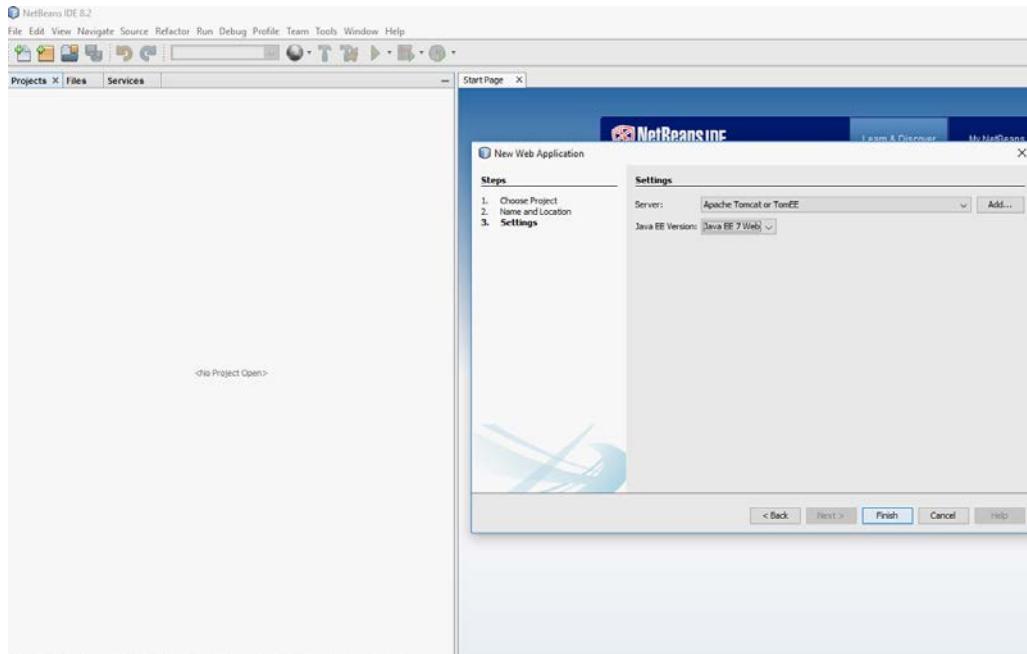


Figura 27. Seleccionando el servidor de aplicación y la versión de Java a utilizar.

Finalmente, se oprime la opción Finish (Terminar) de la pantalla de la figura 27 para concluir el proceso de creación del entorno de desarrollo básico de una aplicación Web Maven en Netbeans, en este caso la aplicación Demo. En la figura 28 se muestra la estructura creada por Netbeans para la aplicación Maven Demo.

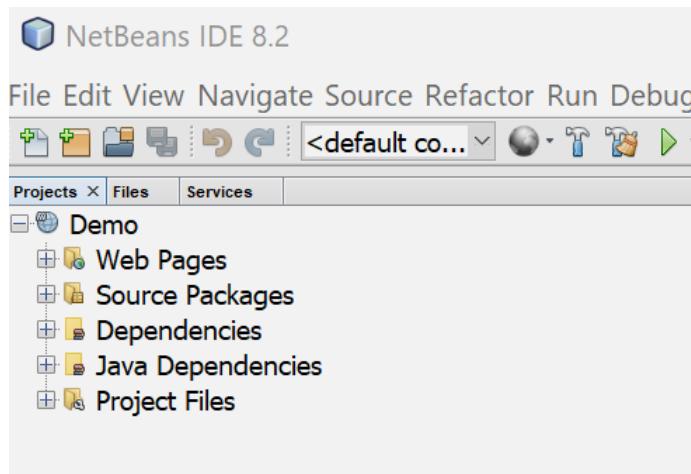


Figura 28. Pantalla de Netbeans con la estructura de un proyecto Maven, en este ejemplo se muestra la estructura de la aplicación Demo.

Observando a detalle la estructura mínima de una aplicación Maven en Netbeans podemos observar que se crean 5 secciones: Web Pages (Páginas Web), Source Packages (Paquetes

Fuentes), Dependencies (Dependencias), Java Dependencies(Dependencias Java) y Project Files (Archivos del Proyecto), ver figura 28.

#### Paso 2.5.2.2: Añadiendo SWBForms y programas auxiliares (dependencias) al proyecto.

Una vez que se ha creado la estructura básica de un proyecto Maven en Netbeans para poder usar **SWBForms** se debe de agregar a la sección Dependencies (Dependencias) los proyectos **SWBDataManager** y **SWBFormsCore** que son el corazón de **SWBForms** y que deben de ser agregados a cada proyecto que quiere hacer uso del conjunto de herramientas de desarrollo **SWBForms**. Estos proyectos pueden ser descargados del repositorio GitHub: <https://github.com/SWBForms> y añadidos de forma similar a los pasos de explicados en el punto 2.4, utilizando las siguientes direcciones web:

- <https://github.com/SWBForms/SWBDataManager.git>
- <https://github.com/SWBForms/SWBFormsCore.git>

Una vez que se realizado las indicaciones de la sección 2.4 para los dos proyectos (**SWBDataManager** y **SWBFormsCore**) se observa una estructura como la que se muestra en la figura 29.

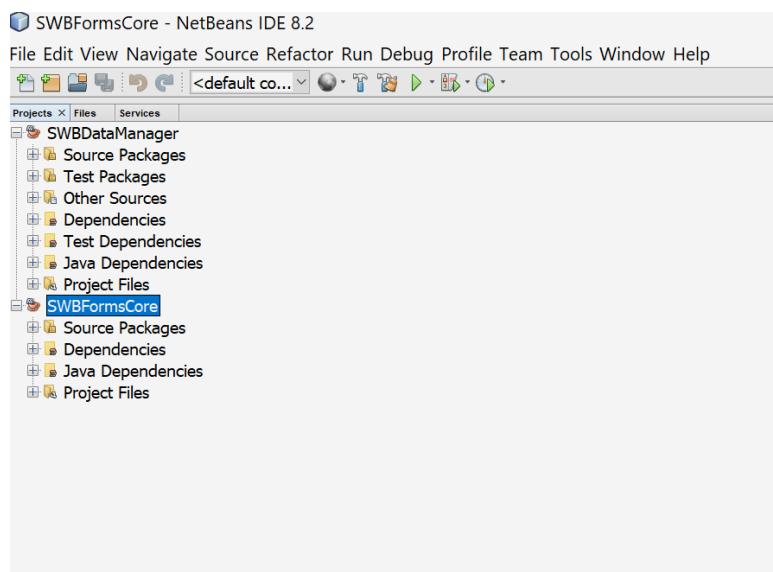


Figura 29. Proyectos **SWBDataManager** y **SWBFormsCore** descargados y abiertos en NetBeans.

Posteriormente se debe incluir ambos proyectos (**SWBDataManager** y **SWBFormsCore**) a la estructura de la aplicación que desea hacer uso de **SWBForms** en este ejemplo la aplicación Demo. En la sección de “Dependencies” (Dependencias) se oprime botón derecho

sobre la palabra “Dependencies” y selecciona la opción Add Dependency (Agrega dependencia), ver figura 30.

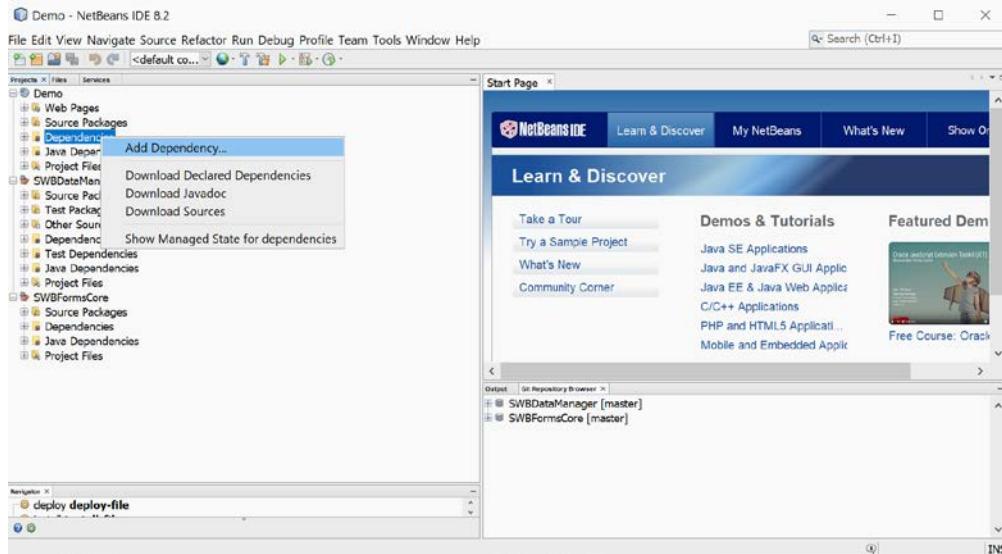


Figura 30. Agregando una dependencia a un proyecto Maven Netbeans.

Una vez seleccionada la opción Add Dependency se muestra una ventana como la que se observa en la figura 31. Seleccionando en esta ventaja la opción “Open Projects” (Proyectos abiertos) se muestran los proyectos abiertos, en este ejemplo SWBDataManager y SWBFormsCore .

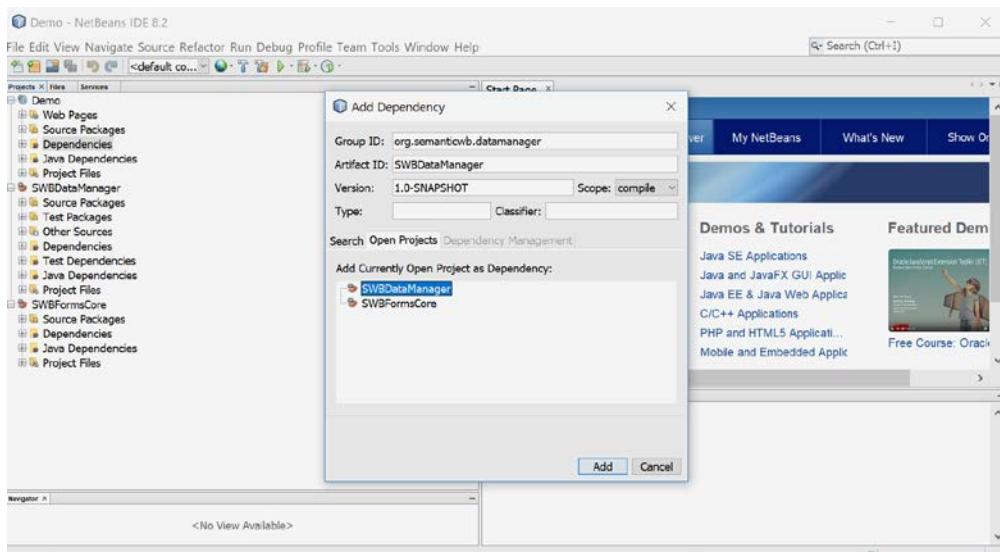


Figura 31. Agregando proyectos como dependencias a otro proyecto Maven en Netbeans.

En la pantalla 31 se selecciona primero el proyecto SWBDataManager y se oprime el botón Add (Adicional) con esto se incluye el proyecto en la estructura de la aplicación Demo como se muestra en la figura 32.

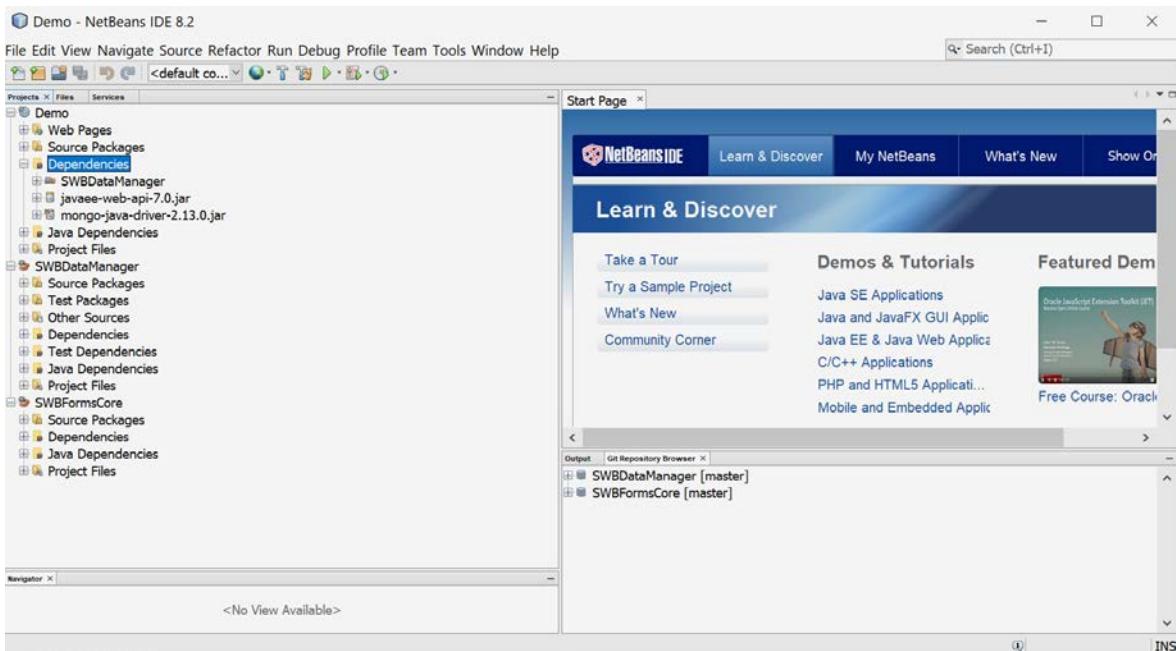


Figura 32. Agregando un proyecto como dependencia a otro proyecto Maven en Netbeans.

Repetiendo el proceso anteriormente descrito se incluye del mismo modo el proyecto SWBFormsCore en la sección dependencias como se muestra en la figura 33.

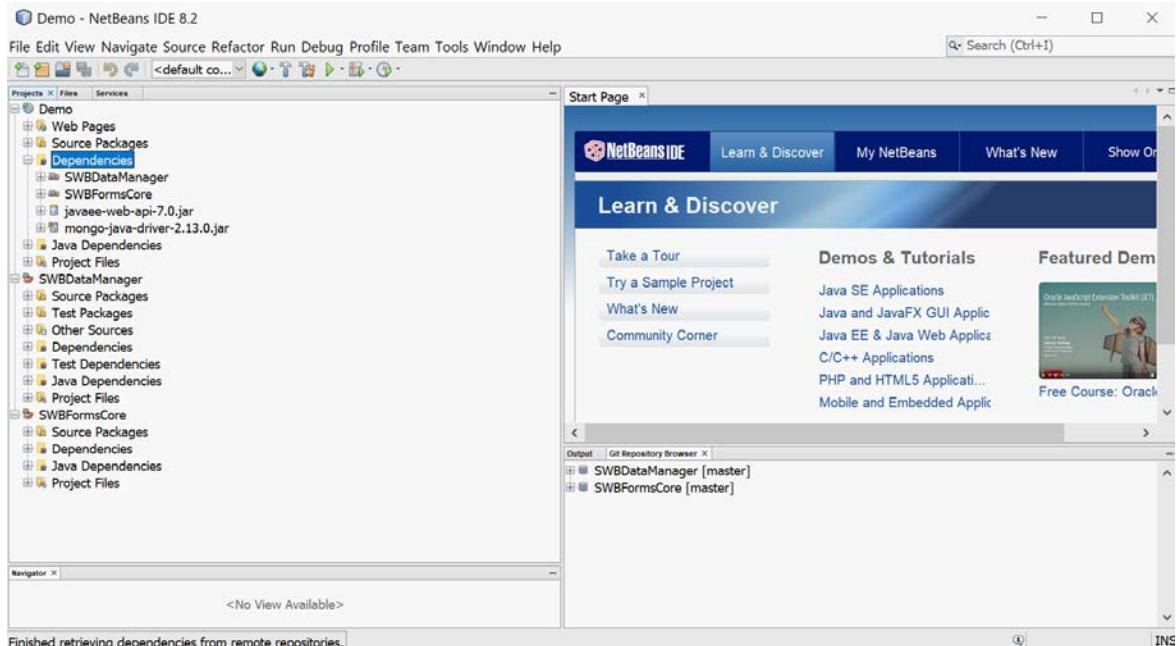


Figura 33. Los proyectos SWBDataManager y SWBFormsCore agregados como dependencias a un proyecto Maven en Netbeans.

Con la estructura que se muestra en la figura 33 no es posible realizar aplicaciones haciendo uso de **SWBForms**, se requiere adicionalmente agregar los siguientes paquetes:

- commons-io-2.2.jar

- javax.mail-1.5.2.jar
- activation-1.1.jar

Para agregar cada uno de los paquetes antes mencionados se debe de seleccionar dentro de la ventana Add Dependency la opción “Search” (Búsqueda) y dentro del campo “Query” (Consulta) se introduce el nombre de paquete a incorporar. En la figura 34 se muestra el resultado de la búsqueda del paquete commons-io.

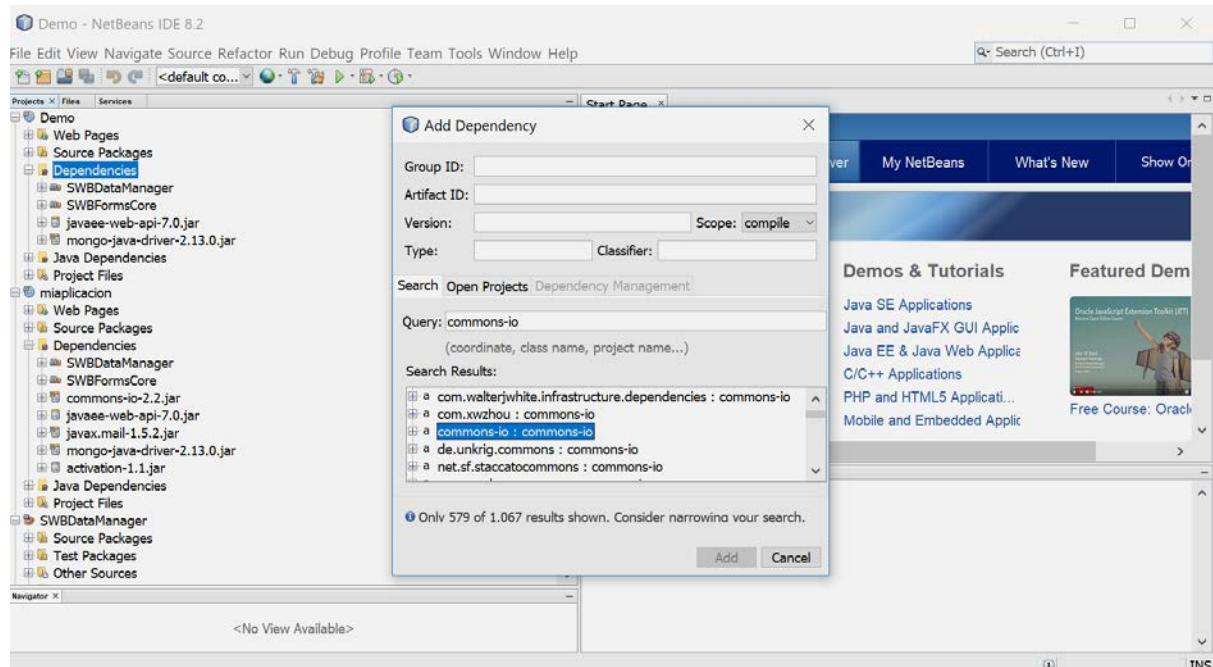


Figura 34. Buscando un paquete a incorporar como dependencia a un proyecto Maven en Netbeans.

Seleccionando la versión a instalar del paquete commons-io explorando dentro de las opciones disponibles mostradas en la figura 25 se oprime el botón adicionar (Add) para que se incluya el paquete como dependencia en el proyecto. Haciendo estos pasos para cada paquete al final la estructura del proyecto Demo queda como se muestra en la figura 35.

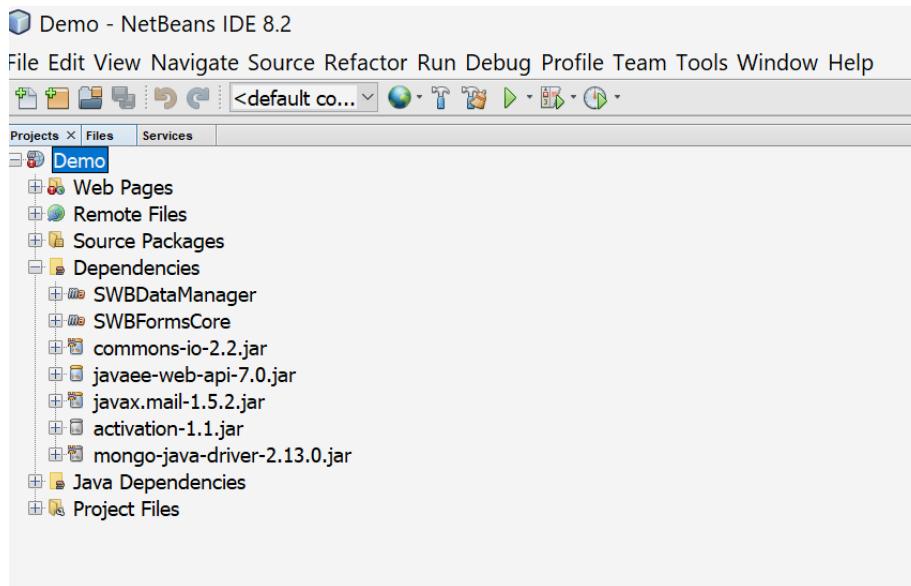


Figura 35. Estructura final de un proyecto Maven en Netbeans que ha adicionado los paquetes y proyectos necesarios para usar **SWBForms** para el desarrollo de aplicaciones web.

Una vez que se ha incluido **SWBForms** al proyecto y paquetes asociados se encuentra ya la estructura mínima de una aplicación en **SWBForms** y esta puede ser ya configurada y desarrollada aprovechando las bondades de **SWBForms** como se verá a partir de la sección 3 del presente manual. Para verificar que se configura de forma adecuada la aplicación para su uso mediante **SWBForms** se puede compilar para ver si no se generan errores. De generarse errores se debe de verificar que no se haya omitido algún paso previo. En la figura 36 se observa que se ha incluido satisfactoriamente **SWBForms** a un proyecto en Netbeans ya que no se generan errores de compilación al ejecutarlo mediante Netbeans.

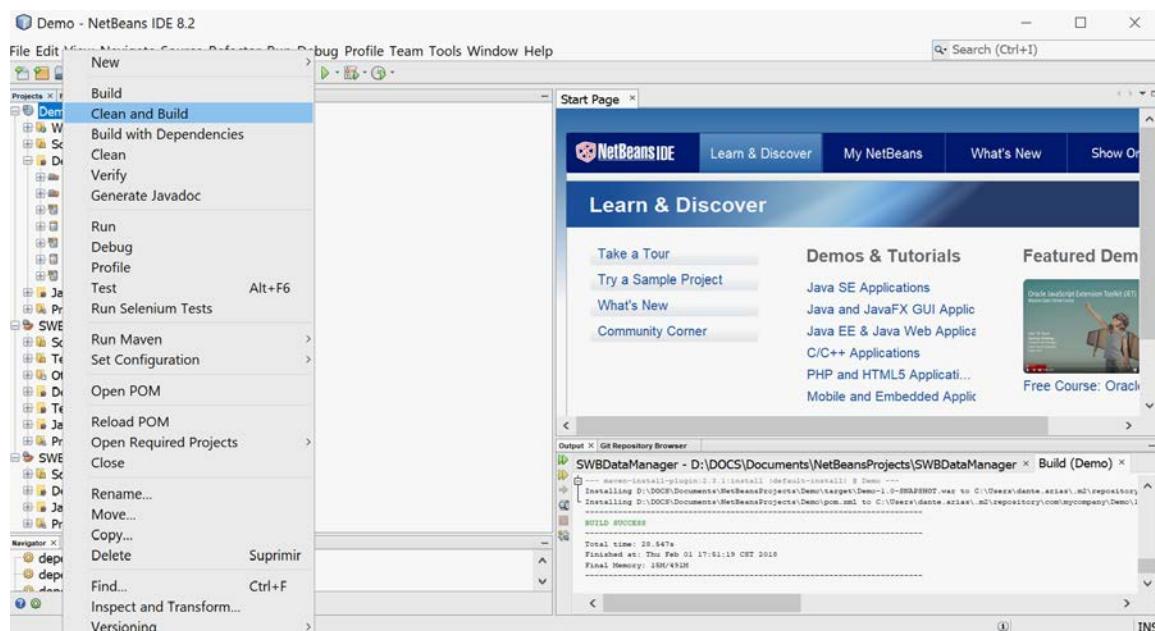


Figura 36. Compilación con éxito de un proyecto en Netbeans que incluye **SWBForms**.

### 3.- Formularios de Datos.

A partir de esta sección se explicarán las capacidades para el desarrollo de formularios de **SWBForms** mediante la explicación del desarrollo de una aplicación Web de una tienda de venta de productos de conveniencia (como una tienda OXXO). En la figura 37 se muestra la pantalla principal de este sistema que servirá de base para mostrar todas las capacidades que SWBForms tiene para el desarrollo de aplicaciones Web.

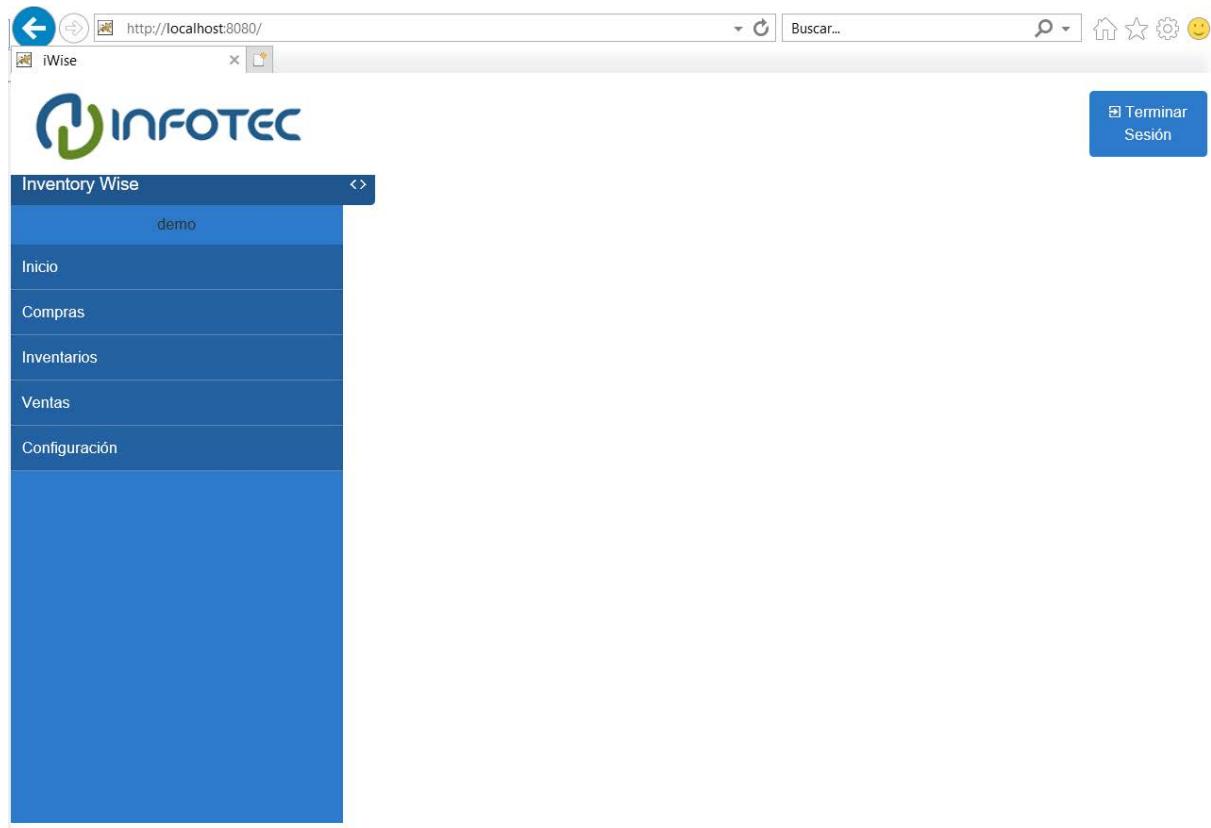


Figura 37. Pantalla principal del sistema de venta de productos de una tienda de conveniencia (aplicación web ejemplo desarrollado en este manual)

Un elemento clave para el desarrollo de aplicaciones es la creación de formularios que permitan al usuario introducir información al sistema.

Figura 38. Formulario para alta de un producto dentro del sistema de ventas de productos por internet de una tienda de conveniencia.

Para el desarrollo de formularios mediante **SWBForms** se debe primero entender que en esta plataforma de desarrollo hay una estrecha relación entre el diseño de un formulario y los datos que procesa por dicho formulario, dichos datos se almacenan en fuentes de datos (datasources). Es así como el primer paso para la creación de un formulario mediante **SWBForms** es la creación de las fuentes de datos. Una fuente de datos es un almacén de datos asociados lógicamente. A nivel de lógica de programación una fuente de datos es una clase de objetos por lo que a lo largo de este manual usaremos indistintamente el término clase o fuente de datos. A nivel de programación se trabaja con clases para el almacenamiento de información en **SWBForms** y a nivel de almacenamiento se trabaja con datos la transformación entre objetos y datos se realiza mediante una capa especial de **SWBForms** conocida como **datamanager**. La creación de una clase para el almacenamiento de datos se realiza en **SWBForms** mediante la instrucción **eng.dataSources**. Todas las definiciones de las fuentes de datos se almacenan en **SWBForms** en un archivo especial llamado **datasources.js**. Por ejemplo, en la aplicación modelo que se explica a lo largo de este manual se requiere almacenar la información de los usuarios del sistema de venta de productos de conveniencia, por lo que es necesario crear una clase para dicha información.

La figura 39 muestra la definición de la clase **Person** que se usa para almacenar la información de los usuarios del sistema.

```
eng.dataSources["Person"] = {
    scls: "Person",
    modelid: "SWBF2",
    datastore: "mongodb",
    displayField: "fullname",
    fields: [
        {name: "firstName", title: "Nombre", type: "string", length: "64", required: true},
        {name: "lastName", title: "Primer Apellido", type: "string", length: "64", required: true},
        {name: "secLastName", title: "Segundo Apellido", type: "string", length: "64", required: false},
        {name: "gender", title: "Género", type: "select", valueMap: "gender", required: true, defaultValue: "male"},
        {name: "birthday", title: "Cumpleaños", type: "date", required: false}
    ]
};
```

Figura 39. Ejemplo de la definición de la clase Person en **SWBForms**

Una vez creada una fuente de datos es posible crear un formulario que procese dichos datos. Los formularios en **SWBForms** se crean como archivos .jsp con una estructura de código mínima como se muestra en el ejemplo de la figura 40. Una aplicación **SWBForms** se compone fundamentales de archivos .jsp y archivos javascript.

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
        <title>JSP Page</title>
    </head>
    <body>
        <h1>Hello World!</h1>
    </body>
</html>
```

Figura 40. Estructura mínima de un documento JSP para la creación de formularios mediante **SWBForms**.

Para comenzar a crear un formulario mediante **SWBForms** se debe de incluir las siguientes líneas de código dentro de la etiqueta <head> de un archivo JSP, ver figura figura 41.

```
<script src="/platform/js/eng.js" type="text/javascript"></script>
<script src="/static/js/f.js" type="text/javascript"></script>
<script type="text/javascript">
    eng.initPlatform("/work/jsp/datasources.js", false);
</script>
```

Figura 41. Etiquetas iniciales para la creación de un formulario en SWBForms.

Las etiquetas de la figura 41 indican por un lado los scripts de referencia para el uso de **SWBForms** y la ubicación del archivo de definición de las fuentes de datos dentro de la estructura de la aplicación que se está desarrollando. En este ejemplo el archivo de configuración de las fuentes de datos (datasources.js) se encuentra en la dirección /work/jsp/datasources.js.

Adicionalmente dentro de la etiqueta <body> del documento JSP se adicionan las siguientes líneas de código, ver figura 42:

```
final DataObject user = (DataObject) session.getAttribute("_USER_");
SWBScriptEngine engine = DataMgr.initPlatform("/work/jsp/datasources.js", session);
final SWBDataSource productDS = engine.getDataSource("Product");
```

Figura 42. Definición de los objetos de control para el manejo de usuarios y fuentes de datos.

Las líneas de código de la figura 42 permiten definir el objeto user para el manejo de las sesiones de usuario, el objeto engine para el manejo de las fuentes de datos y la clase específica que se utilizará en el formulario (clase: Product).

Para ejemplificar cómo se crea un formulario para una fuente de datos, consideremos la creación de un formulario para la alta y edición de los productos que vende una tienda de conveniencia. Para crear este formulario y en general cualquier formulario en **SWBForms** se hace uso de la etiqueta **eng.createForm** dentro de la definición de un script dentro de la etiqueta <body> en el documento JSP como se muestra en la figura 43.

```
<script type="text/javascript">
    var prdId = '<%=suri%>';
    var prdFrm;
    prdFrm = eng.createForm([
        ID:"prdMgr",
        title:"Detalle de Producto",
        top: 10, width: "95%", height: "*",
        numCols: 4,
        titleOrientation: "left",
        showTabs: true,
        autoFocus: true
        ...
        ...
        ...
    ], prdId, "Product");
</script>
```

En esta zona se configura la apariencia y funcionalidad de un formulario.

Los puntos suspensivos indican que pueden haber mas instrucciones dentro de esta sección

Figura 43. Ejemplo de uso de la instrucción eng.createForm para la creación de un formulario en **SWBForms**.

Como se puede observar en la figura 43, la instrucción **eng.createForm** termina en la última instrucción donde se define como parámetro de la instrucción la fuente de datos asociada al formulario en este ejemplo la clase Product.

### 3.1 Etiqueta fields

En la figura 44 se puede observar adicionalmente la operación de la etiqueta **fields** que define qué campos serán visibles dentro de un formulario de los definidos en la fuente de datos asociada al formulario.

```
<script type="text/javascript">
    var prdId = '<%=suri%>';
    var prdFrm;
    prdFrm = eng.createForm({
        ID:"prdMgr"
        ,title:"Detalle de Producto"
        ,top: 10, width: "95%", height: "*"
        ,numCols: 4
        ,titleOrientation: "left"
        ,showTabs: true
        ,autoFocus: true
        ,onLoad: function(form) {
            document.getElementById('xxx').src="/uploadfile/"+form.getField('foto').getValue()[0].id;
        }
        ,baseStyle: "boxedGridCellCyan"
        ,canAdd: true
        ,canEdit: true
        ,canRemove: true
        //,titleSuffix: ":"
        //,requiredTitleSuffix: "<span style='color:#DE3163'>*</span></B>"
        ,fields: [
            {name: "code"}, 
            {name: "productName", width: "100%"}, 
            {name: "description", width: "100%"}, 
            {name: "existence", format: ",0", canEdit: false},
    
```

Figura 44. Uso de campo fields para la definición de los campos a desplegar en un formulario.

Si la etiqueta **fields** es omitida, el formulario tendrá como campos todos los datos definidos en la fuente de datos asociada al formulario, en este ejemplo “Product”, ver figura 43 y 45. En la figura 45 se muestra la definición completa de la clase Product que se ocupa para almacenar los datos de los productos que se comercializan en el sistema de conveniencia que se desarrolla a lo largo de este manual.

```

eng.dataSources["Product"] = {
    scls: "Product",
    modelid: "SWBF2",
    dataStore: "mongodb",
    displayField: "productName",
    fields: [
        {name: "code", title: "Código", required: true, type: "string"},
        {name: "productName", title: "Nombre", required: true, type: "string"},
        {name: "description", title: "Descripción", type: "string"},
        {
            name: "CFDI", title: "Clave CFDI", stype: "select", dataSource: "CFDI", required: true
            ,selectFields: [
                {name: "type"}, {name: "division"}, {name: "group"}, {name: "class"}
            ]
        },
        displayFormat: function(value, record) {
            if(!value) return "";
            var cfdiDS = eng.getDataSource("CFDI");
            var cfdiClassDS = eng.getDataSource("CFDIClass");
            var clss = cfdiClassDS.fetchObjById(value);
            if(clss) {
                return clss.className;
            }
            return value;
        },
        formatCellValue: function(value, record, rowNum, colNum, grid) {
            if(!value) return "";
            var cfdiDS = eng.getDataSource("CFDI");
            var cfdiClassDS = eng.getDataSource("CFDIClass");
            var clss = cfdiClassDS.fetchObjById(value);
            if(clss) {
                return clss.className;
            }
            return value;
        }
    ],
    {name: "existence", title: "Existencia", type: "int", defaultValue: 0},
    {name: "CFDIU", title: "Unidad CFDI", stype: "select", dataSource: "CFDIU", required: true},
    {name: "foto", title: "Foto", stype: "file", required: false },

    {name: "created", title: "Fecha de creación", type: "date", required: false},
    {name: "updated", title: "Fecha de actualización", type: "date", required: false}
]
,links: [
    {name: "optionalData", title: "Datos Opcionales", required: false, stype: "tab", dataSource: "OptionalProductData"},
    {name: "prices", title: "Precios", required: false, stype: "tab", dataSource: "ProductPrice"}
]
};


```

Figura 45. Definición de la fuente de datos Product.

### 3.2 Etiqueta links

La etiqueta links permite asociar diversas fuentes de datos en un mismo formulario, de esta forma por ejemplo con el código de la figura 46 es posible vincular las fuentes de datos Product, OptionalProductData y ProductPrice. Con esta vinculación se puede manejar en distintas fuentes de datos información relacionada como la información de un producto, obteniéndose con esto una mayor flexibilidad en el desarrollo de aplicaciones Web.

```
        ,links: [
            {
                name: "optionalData",
                title: "Datos Opcionales",
                stype: "tab", dataSource: "OptionalProductData",
                numCols: 4,
                titleOrientation: "left",
                fields: [
                    {name: "sku", colspan: 1, width: "100%"},  

                    {name: "iva"},  

                    {name: "volume"},  

                    {name: "weight"},  

                    {name: "category1", startRow: true},  

                    {name: "category2"}
                ]
            },
            {
                name: "prices",
                title: "Precios",
                stype: "tab", dataSource: "ProductPrice",
                //width: 500,
                numCols: 4,
                titleOrientation: "top",
                fields: [
                    {
                        name: "list"
                        ,width: "500"
                        ,fields: [
                            {name: "list", width: "100%"},  

                            {name: "price", width: "100%"}
                        ]
                    }
                ]
            },
        ],
    }, prdId, "Product");
</script>
```

Figura 46. Uso de la etiqueta links para la vinculación de múltiples fuentes de datos

En las figuras 47 y 48 se puede apreciar la integración del código previamente comentado que en su conjunto permite la creación del formulario de la figura 38. Este formulario permite el alta y edición de los datos de los productos que se venden en una tienda de conveniencia.

```
<script type="text/javascript">
    var prdId = '<%=suri%>';
    var prdFrm;
    prdFrm = eng.createForm({
        ID:"prdMgr"
        ,title:"Detalle de Producto"
        ,top: 10, width: "95%", height: "*"
        ,numCols: 4
        ,titleOrientation: "left"
        ,showTabs: true
        ,autoFocus: true
        ,onLoad: function(form) {
            document.getElementById('xxx').src="/uploadfile/"+form.getField('foto').getValue()[0].id;
        }
        ,baseStyle: "boxedGridCellCyan"
        ,canAdd: true
        ,canEdit: true
        ,canRemove: true
        //,titleSuffix: ":"
        //,requiredTitleSuffix: "<span style='color:#DE3163'>*</span>:</B>"
        ,fields: [
            {name: "code"}, 
            {name: "productName", width: "100%"}, 
            {name: "description", width: "100%"}, 
            {name: "existence", format: ",0", canEdit: false}, 
            {name: "CFDI", startRow: true, selectFields: [
                {name: "type"}, 
                {name: "division"}, 
                {name: "group"}, 
                {name: "class"} 
            ]}, 
            {name: "CFDIU", colspan: 1, width: "100%"}, 
            {name: "foto"}, 
            {name: "img", title: "img", type: "img", imageURLPrefix:"/uploadfile/o_1c6svlbauh37tag36cg6vas27"}, 
            {name: "created", useTextField: true, startRow: true, disabled: <%=(suri==null)%>}, 
            {name: "updated", useTextField: true, showIf: "<%=(suri!=null)%>"}
        ]
    })

```

Figura 47. Código completo para la creación del formulario de alta y edición de productos de la tienda de conveniencia, parte 1.

```

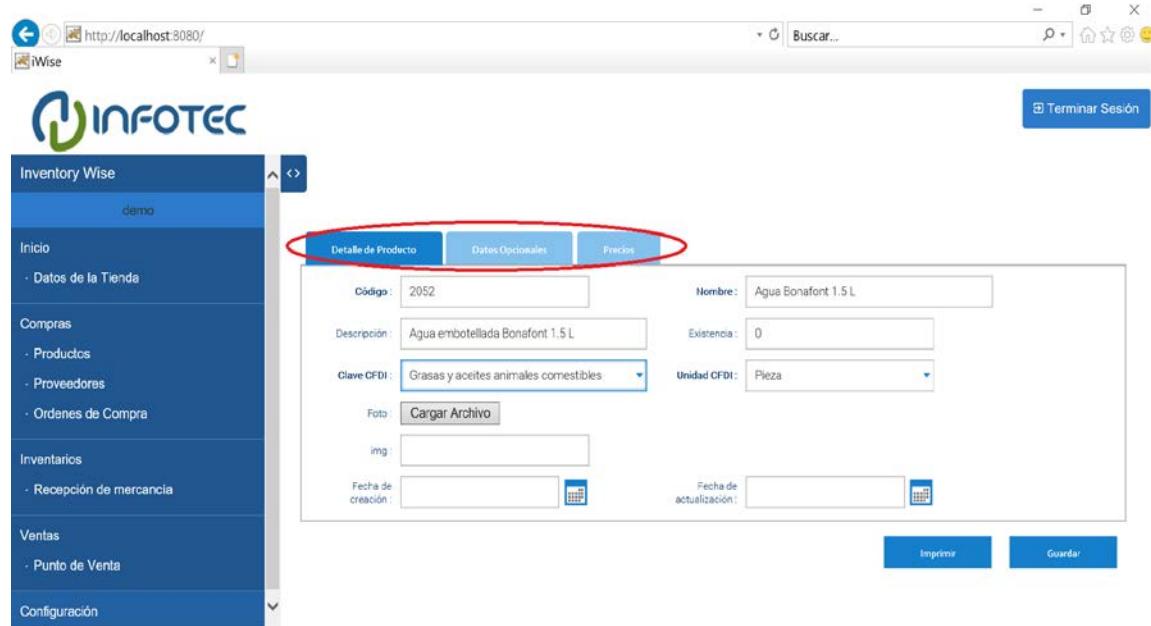
        ,links: [
            {
                name: "optionalData",
                title: "Datos Opcionales",
                stype: "tab", dataSource: "OptionalProductData",
                numCols: 4,
                titleOrientation: "left",
                fields: [
                    {name: "sku", colSpan: 1, width: "100%"},
                    {name: "iva"},
                    {name: "volume"},
                    {name: "weight"},
                    {name: "category1", startRow: true},
                    {name: "category2"}
                ]
            },
            {
                name: "prices",
                title: "Precios",
                stype: "tab", dataSource: "ProductPrice",
                //width: 500,
                numCols: 4,
                titleOrientation: "top",
                fields: [
                    {
                        name: "list"
                        ,width: "500"
                        ,fields: [
                            {name: "list", width: "100%"},
                            {name: "price", width: "100%"}
                        ]
                    }
                ]
            }
        ],
        }, prdId, "Product");
    </script>

```

Figura 48. Código completo para la creación del formulario de alta y edición de productos de la tienda de conveniencia, parte 2.

### 3.3.- stype

En las figuras 45, 47 y 48 podemos observar el uso de la etiqueta **stype** la cual tiene varias funciones dentro de **SWBForms**. La primera de ellas es permitir que se visualicen múltiples formularios dentro de una pantalla mediante pestañas (las pestañas o tabs en inglés son elementos visuales que permiten intercambiar entre múltiples formularios en una misma página, ver figura 49),



The screenshot shows a web browser window with the URL <http://localhost:8080/>. The page title is "iWise". On the left, there is a sidebar menu with categories like Inicio, Compras, Inventarios, Ventas, and Configuración. The main content area displays a product detail form. At the top of the form, there are three tabs: "Detalle de Producto" (selected), "Datos Opcionales", and "Precios". Below the tabs, there are several input fields: Código (2052), Nombre (Aqua Bonafont 1.5 L), Descripción (Aqua embotellada Bonafont 1.5 L), Existencia (0), Clave CFDI (Grasas y aceites animales comestibles), Unidad CFDI (Pieza), Foto (button labeled "Cargar Archivo"), Img (input field), Fecha de creación (input field with calendar icon), and Fecha de actualización (input field with calendar icon). At the bottom right are two buttons: "Imprimir" and "Guardar". The "Detalle de Producto" tab is circled in red.

Figura 49. Múltiples pestañas o tabs en un formulario, se resaltan las pestañas en rojo.

Para desarrollar una pantalla como la que se muestra en la figura 49 que corresponde a la pantalla de alta y modificación de un producto de una tienda de conveniencia se hace uso de una instrucción **stype** como se indica en la figura 50.

```
stype: "tab", dataSource: "OptionalProductData",
```

Figura 50. Instrucción **stype** para la creación de una pestaña dentro de un formulario para el manejo de múltiples formularios como se ilustra en la figura 49

El código completo que permite implementar el formulario mostrado en la figura 49 se muestra en la figura 47 y 48 donde se puede apreciar el uso de la etiqueta **stype: "tab"** en dos ocasiones para generar dos pestañas adicionales que junto con la definición del formulario inicial permiten visualizar 3 formularios en una misma página web.

Otro de los usos de la etiqueta **stype** es permitir que una fuente de datos proporcione datos e información a otro formulario. Esto se realiza mediante la etiqueta **stype: “select”** como se observa en la figura 51.

```

eng.dataSources["CFDI"] = {
    scls: "CFDI",
    modelid: "SWBF2",
    datastore: "mongodb",
    displayField: "class",
    fields: [
        {name: "type", title: "Tipo", required: true, stype: "select", dataSource: "CFDIType"},
        {name: "division", title: "División", required: true, stype: "select", dataSource: "CFDIDivision"},
        {name: "group", title: "Grupo", required: true, stype: "select", dataSource: "CFDIGroup"},
        {name: "class", title: "Clase", required: true, stype: "select", dataSource: "CFDIClass"}
    ]
};

eng.dataSources["CFDIType"] = {
    scls: "CFDIType",
    modelid: "SWBF2",
    datastore: "mongodb",
    displayField: "typeName",
    fields: [
        {name: "typeName", title: "Tipo", required: true, type: "string"}
    ]
};

eng.dataSources["CFDIDivision"] = {
    scls: "CFDIDivision",
    modelid: "SWBF2",
    datastore: "mongodb",
    displayField: "divisionName",
    fields: [
        {name: "divisionName", title: "División", required: true, type: "string"}
    ]
};

eng.dataSources["CFDIGroup"] = {
    scls: "CFDIGroup",
    modelid: "SWBF2",
    datastore: "mongodb",
    displayField: "groupName",
    fields: [
        {name: "groupName", title: "Grupo", required: true, type: "string"}
    ]
};

eng.dataSources["CFDIClass"] = {
    scls: "CFDIClass",
    modelid: "SWBF2",
    datastore: "mongodb",
    displayField: "className",
    fields: [
        {name: "className", title: "Clase", required: true, type: "string"}
    ]
};

```

Figura 51. Definición de la fuente de datos CDFI y sus fuentes asociadas (CFDIType, CFDIDivision, CFDIGroup y CFDIClass) .

Otra de las funciones de la etiqueta **stype** es su uso como se indica en la figura 52 en donde se define la clase ProductPrice y se hace uso de la etiqueta **stype: “grid”** para indicar que la

información de esta fuente de datos se visualice de forma tabular (como una tabla) como se muestra en la figura 53.

```
eng.dataSources["ProductPrice"] = {  
    scls: "ProductPrice",  
    modelid: "SWBF2",  
    dataStore: "mongodb",  
    displayField: "list",  
    fields: [  
        {name: "list", title: "Lista de Precio", required: false, stype: "grid", dataSource: "ProductPriceList"}  
    ]  
};  
  
eng.dataSources["ProductPriceList"] = {  
    scls: "ProductPriceList",  
    modelid: "SWBF2",  
    dataStore: "mongodb",  
    displayField: "list",  
    fields: [  
        {name: "list", title: "Lista de Precio", required: false, stype: "select", dataSource: "PriceList"},  
        {name: "price", title: "Valor", required: false, type: "float"},  
  
        {name: "created", title: "Fecha de creación", type: "date", required: false},  
        {name: "updated", title: "Fecha de actualización", type: "date", required: false}  
    ]  
};
```

Figura 52. Definición de las fuentes de datos ProductPrice y ProductPriceList y el uso de stype: "grid".

En el código de la figura 52 la fuente de datos ProductPrice tiene únicamente un campo llamado list que es alimentado con la información de la fuente de datos ProductPriceList y visualizado en forma de tabla en la figura 53. Como se puede observar únicamente se visualizan los campos Lista de precio y valor ya que en el código de la figura 48 se indican que únicamente se visualicen dichos campos y no todos mediante la etiqueta **fields**.

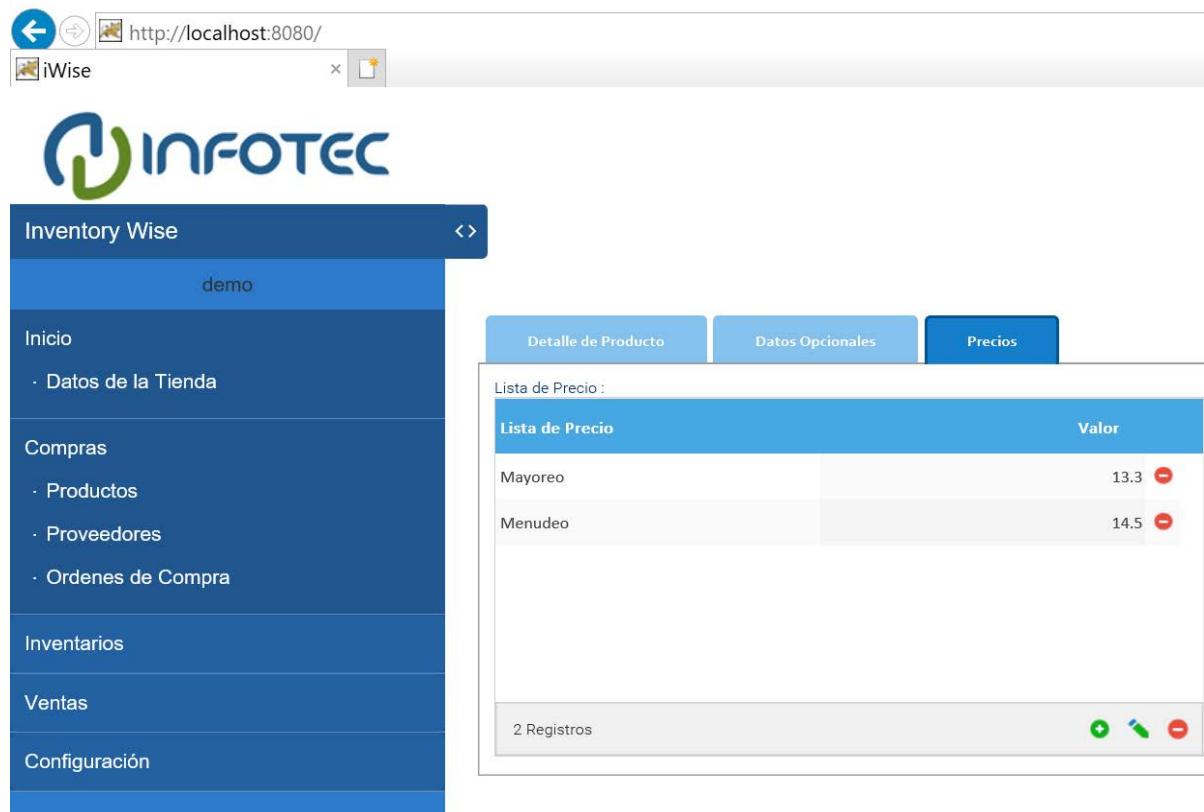


Figura 53. Vista de malla de los datos de una fuente de datos tal y como se indica en el código de la figura 54.

### 3.4.- Tipos de datos

En **SWBForms** se usan los siguientes tipos de datos que se indican mediante la etiqueta **type** como se puede observar por ejemplo el código de la figura 51.

- int
- string
- float
- boolean
- text
- date
- select
- double
- long

### 3.5.- Modificadores de datos

**SWBForms** proporciona los siguientes modificadores para los campos dentro de un formulario a nivel de formato y configuración.

- header
- rowSpacer
- length
- required
- defaultValue
- defaultDynamicValue
- emptyCellValue
- multiple
- multipleAppearance [ picklist | grid]
- disabled
- canEdit
- *readOnly*
- useTextField
- prompt
- mask
- hint
- showHint
- showHintInField [true|false]
- editorType [spinner (SpinnerItem)| TextItem], defaultValue,min,max
- writeStackedIcons [true|false]
- format
- titleOrientation top | left
- startRow true| false
- endRow
- numCols
- colSpan
- width
- autoComplete
- depentSelect
- isUnique
- title

### 3.6.- selectFields

La etiqueta **selectFields** permite indicar que un campo será llenado con la información de una tabla con múltiples de una fuente de datos. Por ejemplo, en la definición de la clase Product que se muestra en la figura 45, podemos observar que se hace uso de la etiqueta **selectFields**, dicha etiqueta que se utiliza en la definición del campo con nombre “CFDI” indica que dicho campo tiene que ser llenado con información de los campos type, division, group y class de la fuente de datos CFDI, ver figura 56.

```
name: "CFDI", title: "Clave CFDI", stype: "select", dataSource: "CFDI", required: true
,selectFields: [
    {name: "type"},
    {name: "division"},
    {name: "group"},
    {name: "class"}
]
```

Figura 56. Asociación entre campos de múltiples fuentes de datos en un formulario.

De esta forma cuando se desee actualizar el valor del campo “CFDI” de la fuente de datos Product en un formulario, los valores posibles se toman de los campos de la fuente de datos CFDI previamente definidos en la figura 51 como se observa en la figura 57.

Foto :	Tipo	División	Grupo	Clase
img :	Productos	Alimentos, Be...	Aceites y grasa...	Grasas y aceit...
Fecha de creación :	Productos	Alimentos, Be...	Bebidas	Bebidas alcoh...
	Productos	Alimentos, Be...	Bebidas	Bebidas no alc...
	Productos	Alimentos, Be...	Bebidas	Café y Té

Figura 57. Efecto visual producido en el formulario Product producido mediante el uso de la etiqueta **selectFields**.

### 3.7.- ShowIf

La etiqueta **showIf** permite establecer restricciones para mostrar o no un campo dentro de un formulario, por ejemplo, en la figura 58 se muestra el uso de la etiqueta **showIf** para limitar la visualización de los campos created y updated. En ambas instrucciones se limita la visualización en los campos respectivos cuando el usuario no se haya autenticado. En cualquier otro caso los campos serían visibles dentro del formulario.

```
{name: "created", useTextField: true, startRow: true, disabled: <%=(suri==null)%>},  
{name: "updated", useTextField: true, showIf: "<%=(suri!=null)%>"}
```

Figura 58. Uso de la etiqueta **showIf** para limitar la visualización de un campo en **SWBForms**.

### 3.8.- Formularios anidados

Otra de las características de SWBForms es la posibilidad de concatenar en una misma pantalla varios formularios de distintas fuentes de datos. Estos formularios incrustados en otro se conocen como subformularios. Bajo este esquema de operación se tiene un formulario principal y dentro de este se definen los subformularios que se deseen ocupar. En la figura 59 se muestra una sección de código que permite el uso de un subformulario dentro de un formulario para el alta y edición de la información sobre proveedores de una tienda de conveniencia.

```
,links: [  
    {  
        name: "address"  
        ,stype: "subForm", dataSource: "Address"  
        ,numCols: 6  
        ,titleOrientation: "left"  
    },
```

Figura 59. Visualización de un subformulario en SWBForms

Como podemos ver en la figura 59 para el manejo de subformularios se hace uso de la etiqueta **stype: "subForm"** donde se indica de donde se toman los datos del subformulario en este ejemplo de la fuente de datos Address. En la figura 60 se muestra el subformulario creado por el código mostrado en la figura 59 y 61.

The screenshot shows a web-based application interface for 'Inventory Wise'. At the top right is a blue button labeled 'Terminar Sesión'. On the left, a vertical sidebar menu includes 'Inicio', 'Compras', 'Productos', 'Proveedores', 'Ordenes de Compra', 'Inventarios', 'Ventas', and 'Configuración'. The main content area has two tabs: 'Detalle de Producto' (selected) and 'Contactos'. The 'Contactos' tab displays a subform with the following fields:

- Razón Social: SeNaSA
- RFC: SEN 551221AB1 ABO 741230ABC
- Sucursal: Tlalnepantla, Edomex
- Teléfono(s): 56242801
- Comentarios: [empty]
- Proveedor desde: [dropdown menu]
- Dirección:
 

Calle:	Obraje	Número exterior:	8	Número interior:	Nave 45
País:		Estado:	Edo. México	Alcaldía/Municipio:	Tlalnepantla
CP:	54000	Colonia/barrio/población:	Industrial		

At the bottom right of the subform area, there is a callout box with the text: 'Esta área es un formulario dentro de otro (subForm)'.

Figura 60. Visualización de un subformulario en **SWBForms**

```

<script type="text/javascript">
    var splrId = '<%=suri%>';
    var splrFrm;
    splrFrm = eng.createForm({
        ID:"splrMgr"
        ,title:"Detalle de Producto"
        ,top: 10, width: "95%", height: "*"
        ,numCols: 4
        ,titleOrientation: "left"
        ,showTabs: true
        ,autoFocus: true
        ,baseStyle: "boxedGridCellCyan"
        ,canAdd: true
        ,canEdit: true
        ,canRemove: true
        ,fields: [
            {name: "businessName", width: "100%"}, 
            {name: "tradeName", width: "100%"}, 
            {
                name: "rfc"
                ,mask: "LLL?000000AAA"
                ,hint: "ABC 741230ABC"
                ,showHintInField: false
            },
            {
                name: "email"
                ,validators: [
                    {stype: "email"}
                ]
            },
            {name: "branchOffice"}, 
            {
                name: "website"
                ,width: "100%"
                ,length: "64"
                ,validators: [
                    {stype: "url"}, 
                    {type: "isUnique", errorMessage: "Ya se tiene registrada esta página web."}
                ]
            },
            {name: "phone"}, 
            {name: "comments", colSpan: 3, width: "100%"}, 
            {name: "created", title: "Proveedor desde", useTextField: true, startRow: true, disabled: <%= (suri==null)%>}
        ]
        ,links: [
            {
                name: "address"
                ,stype: "subForm", dataSource: "Address"
                ,numCols: 6
                ,titleOrientation: "left"
            },
            {
                name: "contacts"
                ,stype: "tab", dataSource: "Contact"
                ,numCols: 4
                ,titleOrientation: "left"
                ,fields: [
                    {name: "fullname", colSpan: 3, width: "100%"}, 
                    {name: "email", width: "100%"}, 
                    {name: "phone", width: "100%"}, 
                    {name: "extension", width: "100%"}
                ]
            }
        ]
    ), splrId, "Supplier");
</script>

```

Figura 61. Código para la creación del formulario que se muestra en la figura 60.

### 3.9.- Ventanas de Edición

Es usual en el diseño de formularios con **SWBForms** el manejo de mallas o tablas como las que se muestran en la figura 62

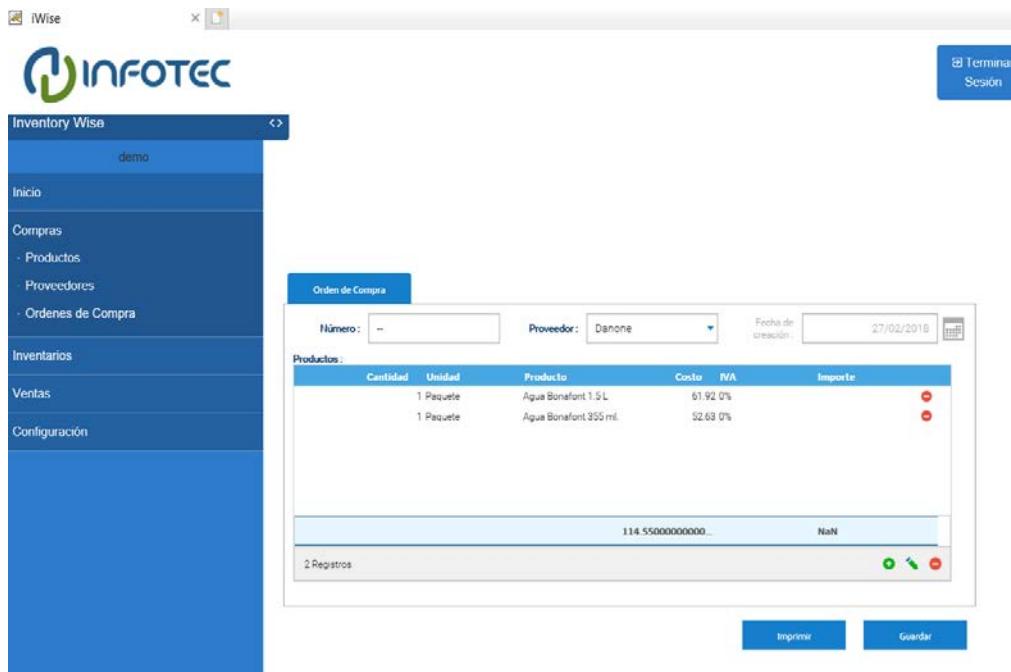


Figura 62. Tabla o malla en un formulario.

En estos casos es posible hacer uso de una ventana de edición que facilite el llenado y/o actualización de los datos de cada fila de la tabla o malla. Esto se puede realizar mediante la etiqueta **winEdit**, la figura 63 muestra un ejemplo del uso de la etiqueta **winEdit**.

```
,winEdit: {
    ID: "item"
    ,title: "Item"
    ,width:"90%", height:"280"
    ,titleOrientation: "left"
    ,showTabs: true
    ,numCols: 6
    ,autoFocus: true
    ,fields: [
        {name: "product", colSpan: 1, width: "100%"},
        {name: "unit", colSpan: 1, width: "100%"},
        {name: "quantity", colSpan: 1, width: "100%"},
        {name: "cost", colSpan: 1, width: "100%"},
        {name: "iva", colSpan: 1, width: "90%"}
    ]
}
```

Figura 63. Ejemplo de uso de la etiqueta **winEdit**.

En la figura 64 se muestra una ventana de edición que se implementa mediante el uso de la instrucción **winEdit**, como se puede observar la ventana de edición se muestra como una ventana emergente que contiene los campos indicados en el código de su definición que se muestra en la figura 63.

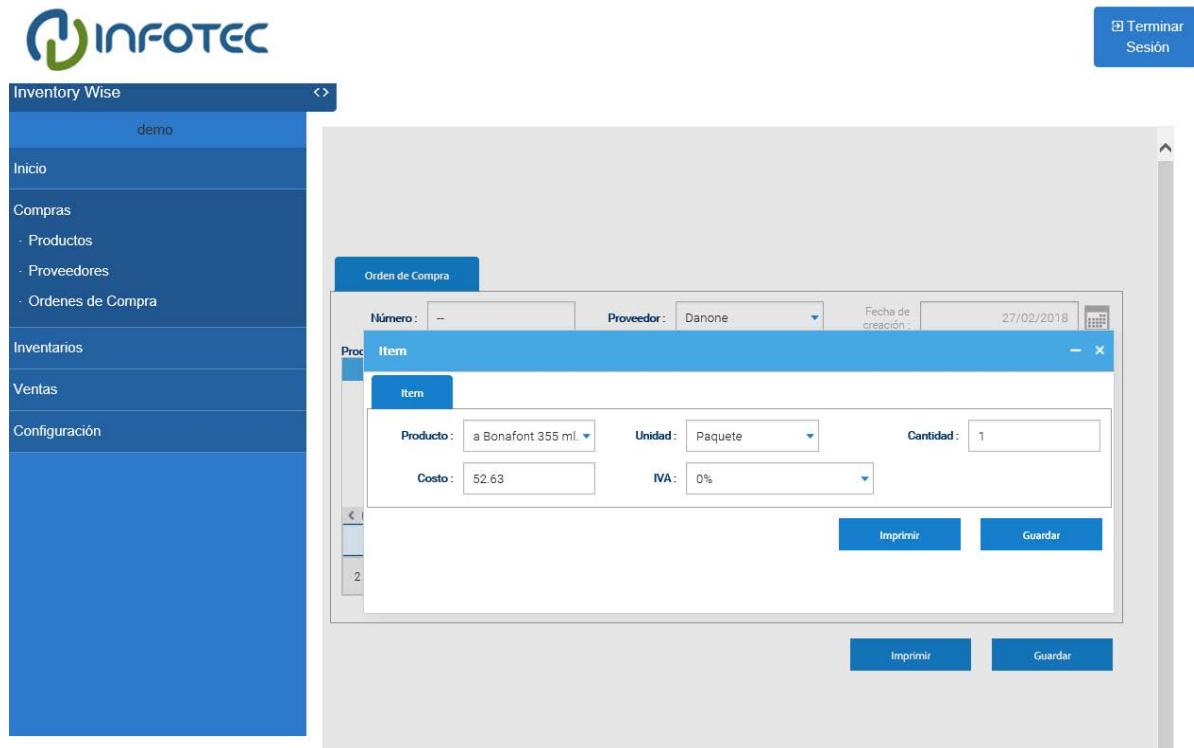


Figura 64. Visualización de una ventana de edición para la generación de las órdenes de compra de nuevos productos para venta de una tienda de conveniencia.

El código completo para la creación del formulario que se muestra en la figura 62 y 64, se puede observar en la figura 65.

```

<script type="text/javascript">
var porderId = '<=&suri%>';
var porderFrm;
porderFrm = eng.createForm({
    ID:"porderMgr"
    ,title:"Orden de Compra"
    ,top: 10, width: "95%", height: "*"
    ,numcols: 6
    ,titleOrientation: "left"
    ,showTabs: true
    ,autoFocus: true
    ,baseStyle: "boxedGridCellCyan"
    ,canAdd: true
    ,canEdit: true
    ,canRemove: true
    //,titleSuffix: ":"
    //,requiredTitleSuffix: "<span style='color:#DE3163'>*</span></B>"
    ,fields: [
        {name: "number", width: "100%", canEdit: false, defaultValue:"--", emptyCellValue: "--"},
        {name: "supplier", width: "100%"}, 
        {name: "created", width: 200, useTextField: true, disabled: true},
        {
            name: "order"
            ,stype: "grid"
            ,dataSource: "ItemOrder"
            ,width:"100%", height:"250", alternateRecordStyles:true, cellHeight:"22"
            ,colspan: 6
            ,titleorientation: "top"
            ,baseStyle: "boxedGridCellRose"
            ,showRollOverCanvas:false
            ,showRollUnderCanvas:false
            ,headerHeight: 22
            //,required: true
            ,showGroupSummary:true
            ,showGridSummary: true
            ,fields: [
                {name: "quantity", width: "150", showGridsummary: false},
                {name: "unit"}, 
                {
                    name: "product", required: true, stype: "select", dataSource: "Product"
                    ,fields: [
                        {name: "code"}, 
                        {name: "productName"}, 
                    ]
                },
                {name: "cost"}, 
                {name: "iva"}, 
                {
                    name: "amount", title: "Importe"
                    ,recordSummaryFunction:"multiplier"
                    ,getGridSummary: function(records, summaryField) {
                        var totalSum=0;
                        for (var i=0; i<records.length; i++) {
                            totalSum+=records[i].amount*records[i].cost;
                        }
                        return totalSum;
                    }
                }
            ] // fields
            ,winEdit: {
                ID: "item"
                ,title: "Item"
                ,width:"90%", height:"280"
                ,titleOrientation: "left"
                ,showTabs: true
                ,numCols: 6
                ,autoFocus: true
                ,fields: [
                    {name: "product", colspan: 1, width: "100%"}, 
                    {name: "unit", colspan: 1, width: "100%"}, 
                    {name: "quantity", colspan: 1, width: "100%"}, 
                    {name: "cost", colspan: 1, width: "100%"}, 
                    {name: "iva", colspan: 1, width: "90%"} 
                ]
            }
        } // order
    ]
}, porderId, "PurchaseOrder");
</script>

```

Figura 65. Código completo para la creación del formulario de las figuras 62 y 64.

### 3.10.- onLoad

En ocasiones es útil cuando se desarrolla una aplicación Web la facilidad ejecutar acciones en el momento en el cual se cargue un formulario, esto se logra en **SWBForms** con la etiqueta **onLoad**. En la figura 66 se muestra un ejemplo del uso de la etiqueta onLoad que permite obtener la información sobre la fotografía de un producto para tenerla la información lista para su despliegue dentro del formulario.

```
,onLoad: function(form) {  
    document.getElementById('xxx').src="/uploadfile/"+form.getField('foto').getValue()[0].id;  
}
```

Figura 66. Ejemplo del uso de la etiqueta onLoad para la ejecución de código al momento de cargar un formulario dentro de una aplicación Web.

### 3.11.- displayFormat y formatCellValue

En ocasiones al desarrollar un formulario se tiene un campo que toma un valor erróneo al momento de ser mostrado, esto es común en la creación de campos en un formato de lista. Por ejemplo, en el formulario de alta y actualización de la información de productos de la aplicación de venta de productos de conveniencia como se observa en la figura 67.

The screenshot shows a web-based form for product management. At the top, there are three tabs: 'Detalle de Producto', 'Datos Opcionales', and 'Precios'. The 'Detalle de Producto' tab is active. The form contains several input fields and dropdown menus. A red oval highlights the 'Clave CFDI' field, which contains the value '\_suri:SWBF2:CFDIClass:5a8db4c257b82df83b33c333'. Below this field is a 'Foto' field with a 'Cargar Archivo' button and an 'img' preview area. Further down are fields for 'Nombre' (Agua Bonafont 1.5 L), 'Descripción' (Agua embotellada Bonafont 1.5 L), 'Existencia' (1), 'Unidad CFDI' (Pieza), 'Fecha de creación' (with a calendar icon), 'Fecha de actualización' (with a calendar icon), and two blue buttons at the bottom labeled 'Imprimir' and 'Guardar'.

Figura 67. Formulario con un valor desplegado erróneo en el campo lista Clave CFDI.

Como se observa dentro del campo Clave CFDI de la figura 67 se muestra dentro del campo una información sobre el nombre de una clase que no corresponde con el valor real esperado. Para solventar este aspecto y colocar dentro de un campo un valor adecuado se utilizan dos etiquetas en **SWBForms**: **displayFormat** y **formatCellValue**. El uso de cada etiqueta dependerá del tipo de formulario sobre el que se esté trabajando, en el caso de un formulario

no tabular como el que se aprecia en la figura 67 se hace uso de la etiqueta **displayFormat**. En la figura 68 se muestra un ejemplo de aplicación de la etiqueta **displayFormat** para desplegar el nombre de la clase CFDI en el título de la lista como es esperado y no el valor que se muestra en la figura 67.

```
,displayFormat: function(value, record) {
    if(!value) return "";
    var cfidiDS = eng.getDataSource("CFDI");
    var cfidiClassDS = eng.getDataSource("CFDIClass");
    var clss = cfidiClassDS.fetchObjById(value);
    if(clss) {
        return clss.className;
    }
    return value;
}
```

Figura 68. Ejemplo del uso de la etiqueta **displayFormat** para corregir el problema mostrado en la figura 67.

Como resultado de la aplicación del código de la figura 68 se muestra en la figura 69 la corrección al problema presentado en la figura 67 para el campo Clave CFDI.

The screenshot shows a web-based form for product details. At the top, there are three tabs: 'Detalle de Producto', 'Datos Opcionales', and 'Precios'. The 'Detalle de Producto' tab is active. Below the tabs, there are several input fields and dropdown menus:

- Código:** 2052
- Nombre:** Agua Bonafont 1.5 L
- Descripción:** Agua embotellada Bonafont 1.5 L
- Existencia:** 1
- Clave CFDI:** Bebidas no alcohólicas (This field is highlighted with a blue border)
- Unidad CFDI:** Pieza
- Foto:** Cargar Archivo
- img:** (Empty input field)
- Fecha de creación:** (Empty input field) with a calendar icon
- Fecha de actualización:** (Empty input field) with a calendar icon

At the bottom right of the form are two buttons: 'Imprimir' and 'Guardar'.

Figura 69. Corrección al campo Clave CFDI producto del código de la figura 68.

Por su parte la etiqueta **formatCellValue** al igual que la etiqueta **displayFormat** permite corregir el llenado de valores en campos que son leídos desde una fuente de datos pero a diferencia de la etiqueta **displayFormat**, en el caso de la etiqueta **formatCellValue** se usa en el caso de un formulario en su representación tabular como el que se muestra en la figura 70.

Código	Nombre	Descripción	Clave CFDI	Existencia
2052	Agua Bonafont 1.5 L	Agua embotellada Bonafont 1.5_l_suri:SWBF2:CFDIClass:5a&db-	1	1
2050	Agua Bonafont 355 ml.	Agua embotellada Bonafont 355_ml_suri:SWBF2:CFDIClass:5a&db-	5	5
2051	Agua Bonafont 600 ml.	Agua embotellada Bonafont 600_ml_suri:SWBF2:CFDIClass:5a&db-	4	4

Figura 70. Ejemplo de un formulario en su representación tabular que muestra el listado de productos de una tienda de conveniencia.

Como se puede observar en la figura 70 el campo clave CFDI tiene valores que no corresponden con los valores correctos para este campo, para corregir esta problemática se debe de utilizar la etiqueta **formatCellValue** como se muestra en la figura 71.

```

,formatCellValue:
    function(value, record, rowNum, colNum, grid) {
        if(!value) return "";
        var cfдиDS = eng.getDataSource("CFDI");
        var cfдиClassDS = eng.getDataSource("CFDIClass");
        var clss = cfдиClassDS.fetchObjById(value);
        if(clss) {
            return clss.className;
        }
        return value;
    }
}

```

Figura 71. Ejemplo del uso de la etiqueta **formatCellValue** para corregir el problema mostrado en la figura 70.

Como resultado de la aplicación del código de la figura 71 se muestra en la figura 72 la corrección al problema presentado en la figura 70 para el campo Clave CFDI.

The screenshot shows the Inventory Wise application interface. On the left is a sidebar with navigation links: Inicio, Compras (Productos, Proveedores, Órdenes de Compra), Inventarios, Ventas, and Configuración. The main area displays a table titled 'demo' with columns: Código, Nombre, Descripción, Clave CFDI, and Existencia. The table contains three rows of data:

Código	Nombre	Descripción	Clave CFDI	Existencia
2052	Agua Bonafont 1.5 L	Agua embotellada Bonafont 1.5 Bebidas no alcohólicas		1
2050	Agua Bonafont 355 ml.	Agua embotellada Bonafont 355Bebidas no alcohólicas		5
2051	Agua Bonafont 600 ml.	Agua embotellada Bonafont 600Bebidas no alcohólicas		4

At the bottom of the table, it says '3 Registros'. The application has a header with the INFOTEC logo and a 'Terminar Sesión' button.

Figura 72. Corrección al campo Clave CFDI producto del código de la figura 71.

## 4.- Organizar la apariencia de los formularios

**SWBForms** proporciona diversas instrucciones o etiquetas que permiten adaptar la apariencia y funcionalidad de los campos en un formulario de acuerdo a las necesidades de cada aplicación. A continuación, se explican dichas instrucciones.

### 4.1.- validator

La etiqueta **eng.validator** permite definir restricciones en los valores y el formato de los valores que un usuario puede introducir en un campo. Este tipo de restricciones se conocen como **validators** en **SWBForms**. Por ejemplo, en la figura 73 se muestra un ejemplo del uso de la etiqueta **eng.validators** que se aplica para limitar los valores posibles para un campo de correo electrónico y define un mensaje de error si el usuario no cumple con el formato y valores de un correo electrónico. Para indicar los valores y el formato de un campo se hace uso de expresiones regulares.

```
eng.validators["email"] = {type:"regexp",
    expression:"^([a-zA-Z0-9_.\\-]+@[a-zA-Z0-9\\-]+\\.)+[a-zA-Z0-9]{2,4}$",
    errorMessage:"No es un correo electrónico válido"};
```

Figura 73. Ejemplo de uso de la etiqueta **eng.validators**.

Una vez definido un validador este puede ser usado en cualquier definición de un campo como se muestra en la figura 74, en donde se asocia el validador creado con la instrucción de la figura 73 al campo email de la clase Clientes.

```
eng.dataSources["Clientes"] = {
    scls: "Clientes",
    modelid: "SWBF2",
    dataStore: "mongodb",
    displayField: "nombre",
    fields: [
        {name: "nombre", title: "Nombre", required: true, type: "string"},
        {name: "email", title: "Correo Electrónico", required: true, type: "string", validators:[{stype:"email"}]},
        {name: "nacimiento", title: "Fecha nacimiento", type: "date"},  

        {name: "direccion", title: "Dirección", type: "text"},  

        {name: "telefono", title: "Telefono", type: "string"},  

        {name: "rfc", title: "RFC", required: false, type: "string"},  

        {name: "sexo", title: "Sexo", stype: "select",
            valueMap:{male:"Hombre",female:"Mujer"}},
```

Figura 74. Aplicación de validador (validator) en la definición de una clase en **SWBForms**.

#### 4.2.- startRow

La etiqueta **startRow** permite alterar el orden en el cual aparecen los campos en un formulario. Por defecto los campos de un formulario se muestran de forma consecutiva de derecha a izquierda respetando el número de columnas definidas para un formulario (el manejo de columnas se verá a detalle mediante el manejo de la etiqueta **numCols**), si queremos que un campo no respete este comportamiento podemos usar la etiqueta **startRow** que indica que los campos que aún no se han mostrado inicien en una nueva “fila” del formulario. En la figura 75 se muestra el uso de la etiqueta **StartRow** en la definición de un formulario en **SWBForms**. Los valores que pueden tomar esta etiqueta son **true** o **false** para activar o desactivar el efecto de la etiqueta.

```
{  
    name: "CFDI"  
    ,startRow: true  
    ,selectFields: [  
        {name: "type"},  
        {name: "division"},  
        {name: "group"},  
        {name: "class"}  
    ]  
    ,colSpan: 1  
    ,width: "100%"  
},
```

Figura 75. Ejemplo del uso de la etiqueta **startRow** en la definición de un formulario.

En la figura 76 y 77 se muestra el efecto del uso de la etiqueta **startRow**. En la figura 76 se muestra la distribución por defecto de los campos en un formulario. Como podemos observar el campo **Clave CFDI** se muestra a la izquierda del campo **Existencia**. Una vez que se asigna el valor **true** a la etiqueta **startRow** como se indica en la figura 75 se obtiene una nueva distribución de los campos comenzando a partir del campo **Clave CFDI** como se observa en la figura 77. Como se puede observar ahora el campo **Clave CFDI** se muestra al inicio de una nueva línea de campos dentro del formulario.

Inventory Wise

demo

Inicio

Compras

- Productos
- Proveedores
- Ordenes de Compra

Inventarios

Ventas

Configuración

Detalle de Producto

Datos Opcionales

Precios

Código: 2052 Nombre: Agua Bonafont 1.5 L Descripción: Agua embotellada Bon

Existencia: 0 Clave CFDI: Bebidas no alcohólicas Unidad CFDI: Pieza

Foto: Cargar Archivo

img:

Fecha de creación: Fecha de actualización:

Imprimir Guardar

Figura 76. Configuración por defecto para la visualización de los campos de un formulario.

Inventory Wise

demo

Inicio

Compras

- Productos
- Proveedores
- Ordenes de Compra

Inventarios

Ventas

Configuración

Detalle de Producto

Datos Opcionales

Precios

Código: 2052 Nombre: Agua Bonafont 1.5 L Descripción: Agua embotellada Bon

Existencia: 0 Clave CFDI: Bebidas no alcohólicas Unidad CFDI: Pieza

Foto: Cargar Archivo

img:

Fecha de creación: Fecha de actualización:

Imprimir Guardar

Figura 77. Efecto del uso de la etiqueta startRow.

#### 4.3.-width

La etiqueta **width** permite definir la longitud de campos y formularios dentro de **SWBForms**. Los valores asociados a este campo son en porcentajes como se observa en la figura 78 en donde se ocupa la etiqueta **width** para indicar la longitud de despliegue de un formulario. El porcentaje indicada la proporción que ocupará un formulario respecto al área disponible para el despliegue del mismo dentro de una página web.

```
prdFrm = eng.createForm({
    ID:"prdMgr"
    ,title:"Detalle de Producto"
    ,top: 10, width: "95%", height: "*"
    ,numCols: 4
    ,titleOrientation: "left"
    ,showTabs: true
    ,autoFocus: true
    ,onLoad: function(form) {
        document.getElementById('xxx').src="/uploadfile/"+form.getField('foto').getValue()[0].id;
    }
    ,baseStyle: "boxedGridCellCyan"
    ,canAdd: true
    ,canEdit: true
    ,canRemove: true
});
```

Figura 78. Ejemplo del uso de la etiqueta **width**.

Como se puede observar en las figuras 79 y 80 el ancho de un formulario cambia producto del efecto de la etiqueta **width** dentro de la definición de un formulario.

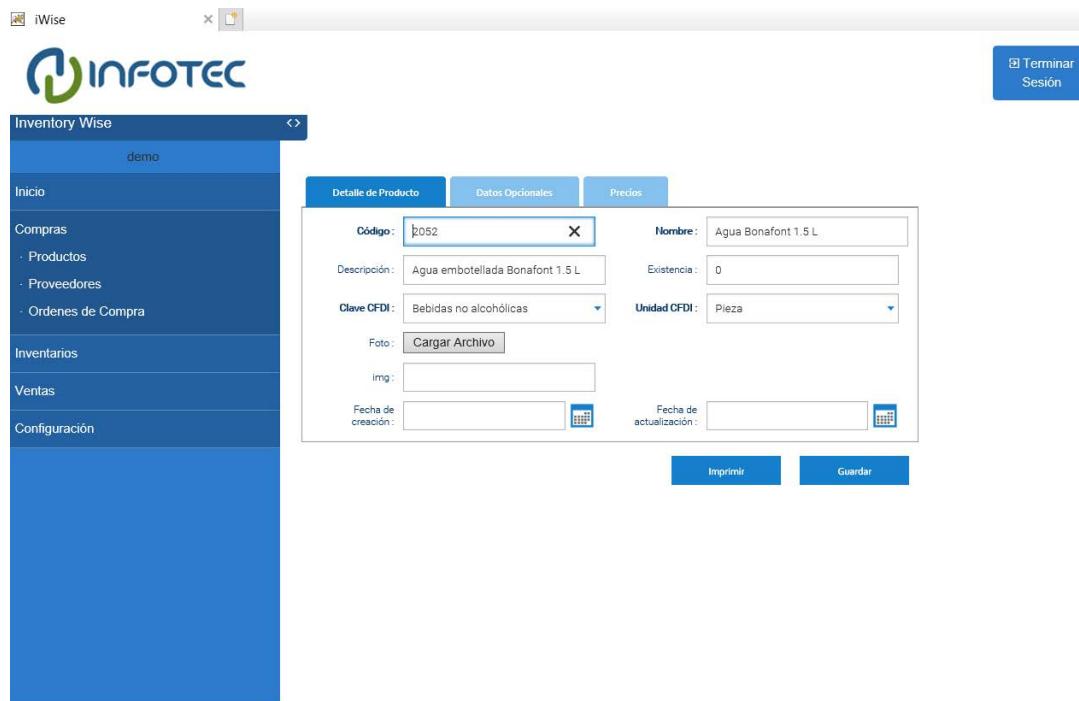


Figura 79. Configuración inicial de despliegue de un formulario. El porcentaje de despliegue del formulario es del 80% del área disponible dentro de la página web.

Figura 80. Configuración final de despliegue del formulario de la figura 72 después de actualizar el valor de la etiqueta **width** a 95%.

Un efecto similar al producido en un formulario tiene la etiqueta **width** cuando se usa en la definición de un campo dentro de un formulario como se muestra en la figura 81.

```
{name: "productName", width: "100%"},  
 {name: "description", width: "100%"},
```

Figura 81. Ejemplo del uso de la etiqueta **width** en la definición de un campo

En la figura 82 se puede ver el efecto de la aplicación de los códigos de la figura 81. Como se puede observar los campos Nombre (productName) y Descripción (description) abarcan todo el espacio disponible dentro de cada “fila” y “columna” respectiva.

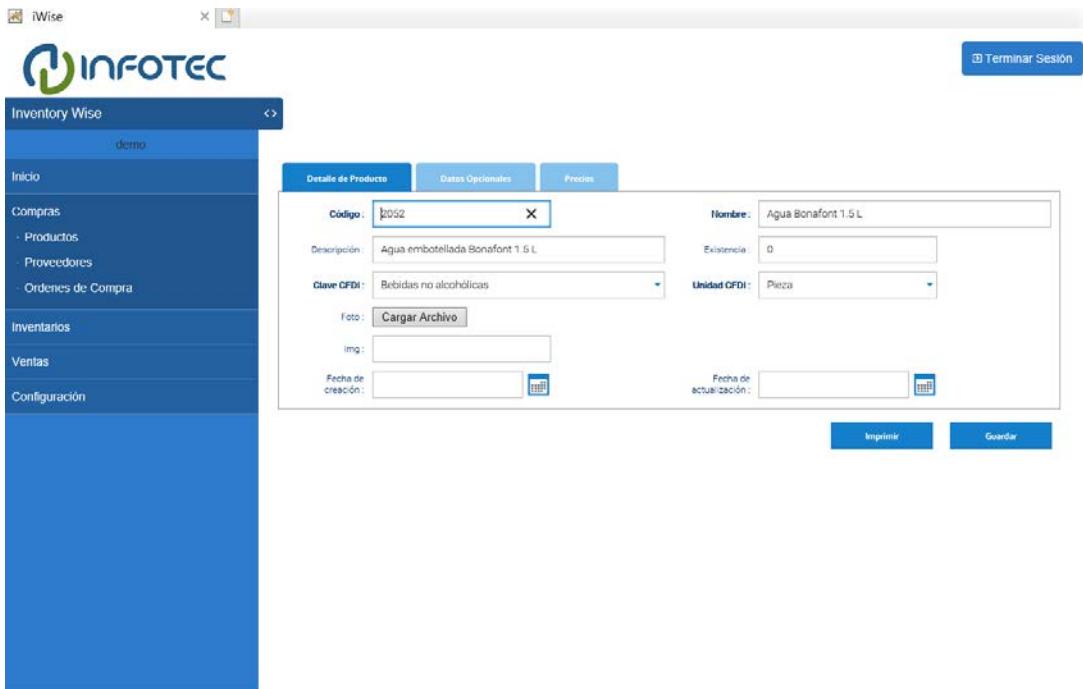


Figura 82. Efecto de la aplicación de la etiqueta **width** sobre los campos Nombre y Descripción según el código de la figura 81.

En la figura 83 se muestra ahora un cambio en la longitud del campo Descripción (description), ajustando su tamaño a un 50% del área disponible de despliegue.

```
{name: "productName", width: "100%"},  
{name: "description", width: "50%"},
```

Figura 83. Ajuste de la longitud de un campo mediante la etiqueta **width**.

En la figura 84 se muestra el efecto del ajuste en la longitud del campo Descripción (description) como se indica en el código de la figura 83. Como se puede apreciar el campo Descripción disminuye su longitud en un 50%.

The screenshot shows a software interface titled "Inventory Wise" with a blue header bar. On the left, there's a sidebar with menu items: "Inicio", "Compras" (with sub-options "Productos", "Proveedores", "Ordenes de Compra"), "Inventarios", "Ventas", and "Configuración". The main area has a title "Detalle de Producto" and tabs "Datos Opcionales" and "Precios". The form contains fields for "Código" (2052), "Nombre" (Agua Bonafont 1.5 L), "Descripción" (Agua embotellada Bonafont), "Existencia" (0), "Clave CFDI" (Bebidas no alcohólicas), "Unidad CFDI" (Pieza), "Foto" (button to "Cargar Archivo"), "img" (empty input), "Fecha de creación" (empty input with calendar icon), and "Fecha de actualización" (empty input with calendar icon). At the bottom are "Imprimir" and "Guardar" buttons.

Figura 84. Ajuste en la longitud del campo Descripción según el código de la figura 83.

#### 4.4.-numCols

La etiqueta **numCols** permite definir el número de columnas virtuales en las que se divide un formulario. En la figura 85 se muestra el efecto de asignar a la etiqueta **numCols** el valor 4. Cada cuadrado resalta una columna virtual dentro de un formulario. Por defecto el número de columnas de un formulario es de 6 columnas.

This screenshot is identical to Figure 84, but it illustrates the effect of setting the **numCols** attribute to 4. The form is now divided into four equal-width columns by vertical grid lines. The fields "Código", "Nombre", "Descripción", and "Existencia" are in the first column; "Clave CFDI", "Unidad CFDI", and "Foto" are in the second column; "img", "Fecha de creación", and "Fecha de actualización" are in the third column; and the buttons "Imprimir" and "Guardar" are in the fourth column. The rest of the interface, including the sidebar and tabs, remains the same.

Figura 85. Un formulario con 4 columnas.

En la figura 86 se muestra el efecto de asignar a la etiqueta **numCols** el valor 6 para que el formulario tenga 6 columnas virtuales. Como se puede observar la distribución de los campos cambia respecto a la distribución de la figura 85.

The screenshot shows a web-based application interface titled "Inventory Wise" with the sub-page "demo". On the left, there is a sidebar with navigation links: Inicio, Compras (Productos, Proveedores, Ordenes de Compra), Inventarios, Ventas, and Configuración. The main content area is titled "Detalle de Producto" and contains the following fields arranged in 6 columns:

- Código:  (Column 1)
- Nombre:  (Column 2)
- Descripción:  (Column 3)
- Existencia:  (Column 4)
- Clave CFDI:  (Column 5)
- Unidad CFDI:  (Column 6)
- Foto:  (Column 1)
- Img:  (Column 2)
- Fecha de creación:  (Column 3)
- Fecha de actualización:  (Column 4)

At the bottom right of the form are two buttons: "Imprimir" and "Guardar". In the top right corner of the application window, there is a blue button labeled "Terminar Sesión".

Figura 86. Un formulario con 6 columnas.

#### 4.5.- titleOrientation

La etiqueta **titleOrientation** permite definir donde se visualizará el título asociado a cada campo de un formulario, en la figura 87 se muestra un ejemplo del uso de la etiqueta **titleOrientation**.

```
,titleOrientation: "left"
```

Figura 87. Ejemplo del uso de la etiqueta **titleOrientation**.

En este contexto el título de un campo es el texto asociado a cada campo, un ejemplo se resalta mediante un círculo rojo en la figura 88. Como se puede observar en la figura 88, los títulos de todos los campos aparecen a la izquierda de los campos. Para el logro de este efecto visual se debe de asignar a la etiqueta **titleOrientation** el valor “left” como se muestra en la figura 87.

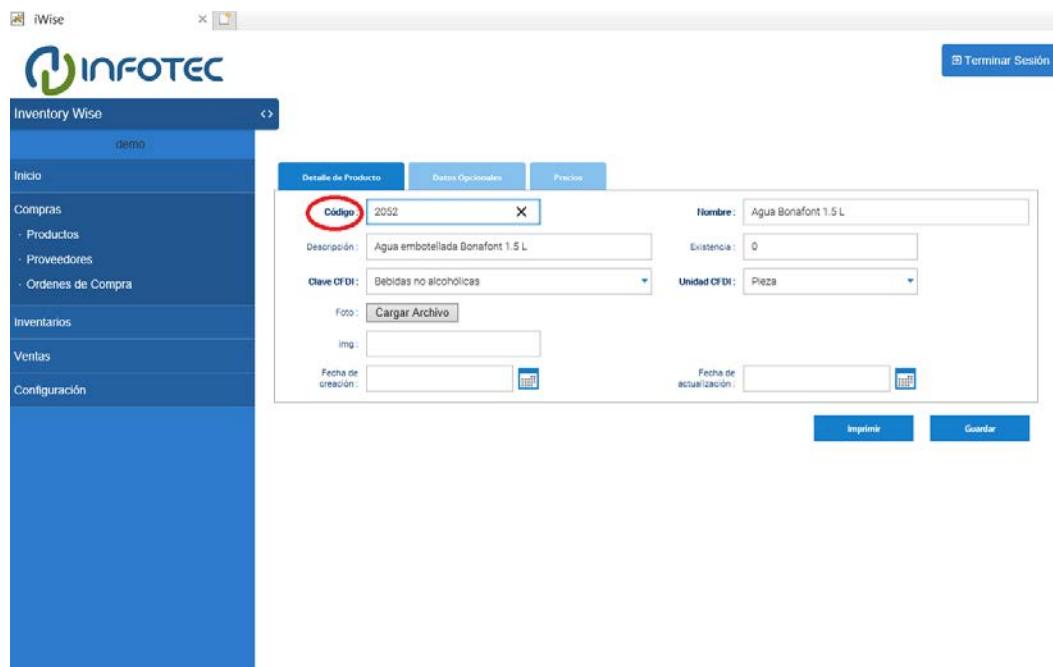


Figura 88. Títulos de campos alineados a la izquierda.

Ahora bien si se asigna el valor “right” a la etiqueta **titleOrientation** como se muestra en la figura 89 se consigue que todos los títulos de los campos de un formulario aparezcan a la derecha de los campos.

```
, titleOrientation: "right"
```

Figura 89. Etiqueta **titleOrientation** para alinear a la derecha de los títulos de los campos de un formulario.

Como se puede observar en la figura 90 las etiquetas de los campos están alineadas a la derecha de los mismos producto de utilizar el código de la figura 89.

Figura 90. Formulario con la alineación de los títulos de los campos a la derecha.

#### 4.6.-mask

Una forma adicional de limitar los posibles valores de un campo dentro de un formulario es el uso de la etiqueta **mask** como se muestra en la figura 84. En el código de la figura 84, la etiqueta **mask** permite definir una regla o máscara para el campo rfc. La regla definida se indica entre comillas siguiendo los estándares definidos en la tabla siguiente:

Carácter	Significado
0	Dígito (0 al 9) o los signos más(+) o menos(-)
9	Dígito o espacio
#	Dígito
L	Letra (A a la Z)
?	Letra (A a la Z) o espacio
A	Letra o dígito
a	Letra o dígito
C	Cualquier carácter o espacio
<	Todos los caracteres a partir del signo se convierten en minúsculas
>	Todos los caracteres a partir del signo se convierten en mayúsculas

Tabla 1. Representación de información mediante máscaras en **SWBForms**.<sup>1</sup>

<sup>1</sup> [https://www.smartclient.com/smartgwt/showcase/#form\\_masking](https://www.smartclient.com/smartgwt/showcase/#form_masking)

Haciendo uso de máscaras se limita las posibles combinaciones de valores que se pueden ingresar al campo, en el ejemplo de la figura 91 se indica mediante la etiqueta **mask** la estructura de un rfc válido.

```
{  
    name: "rfc"  
    ,mask: "LLL?000000AAA"  
    ,hint: "ABC 741230ABC"  
    ,showHintInField: false  
},
```

Figura 91. Ejemplo del uso de la etiqueta **mask**

#### 4.7.- **hint**

La etiqueta **hint** nos permite mostrar en pantalla un texto de ayuda o explicativo para que el usuario pueda introducir valores válidos o aceptables para un campo. En el código de la figura 91 se muestra como se usa la etiqueta **hint** para indicar un texto de ayuda para el llenado del campo rfc de tal forma que los datos introducidos sean válidos. Visualmente por defecto se muestra el texto de ayuda indicado en la etiqueta **hint** a la izquierda del campo y fuera del mismo como se muestra en la figura 92.

The screenshot shows a web-based application interface. On the left, there's a sidebar with navigation links like 'Inicio', 'Compras', 'Inventarios', 'Ventas', and 'Configuración'. The main content area has tabs for 'Proveedores' and 'Clientes'. The 'Proveedores' tab is active. A form is displayed with various fields:  
- Razón Social: SeñalSA  
- RFC: ABC 741230ABC (highlighted with a red circle)  
- Sucursal: Tlalnepantla, Edomex  
- Teléfono(s): 56242801  
- Comentarios:  
- Proveedor desde:  
- Dirección:  
 - Calle: Obraje  
 - Número exterior: 8  
 - Número interior: Nave 45-A  
 - País:  
 - Estado: Edo. México  
 - CP: 54000  
 - Colonia/barrio/población: Industrial  
At the bottom right of the form are 'Imprimir' and 'Guardar' buttons.

Figura 92. Ejemplo del uso de la etiqueta **hint** para el campo rfc dentro de un formulario de acuerdo al código de la figura 91.

#### 4.8.- showHintInField

Asociado al uso de la etiqueta **hint** se hace uso de la etiqueta **showHintInField** para indicar si se desea que el mensaje de ayuda de la etiqueta **hint** se muestra fuera o no del campo. Por defecto el mensaje que se indica en la etiqueta **hint** se muestra afuera del campo como se indica en la figura en 92. Mediante el uso del código de la figura 93 se logra que el texto de ayuda de la etiqueta **hint** se muestra dentro del campo producto del uso de la etiqueta **showHintInField** con valor **true**.

```
{  
    name: "rfc"  
    ,mask: "LLL?000000AAA"  
    ,hint: "ABC 741230ABC"  
    ,showHintInField: true  
},
```

Figura 93. Ejemplo de uso de la etiqueta **showHintInField**.

Como se muestra en la figura 94 el texto de ayuda indicado en la etiqueta **hint** del campo rfc se muestra dentro del campo producto del uso de la etiqueta **showHintInField** como se indica en el código de la figura 93.

The screenshot shows a web-based inventory management system. On the left, there's a sidebar with navigation links like 'Inicio', 'Compras', 'Sucursales', 'Ventas', and 'Configuración'. The main content area has tabs for 'Proveedores' and 'Contactos', with 'Proveedores' currently selected. The form fields include:  
- Razón Social: SeNaISA  
- RFC: ABC 741230ABC (with help text 'ABC 741230ABC' shown inside the input field)  
- Sucursal: Tlalnepantla, EdoMex  
- Teléfono(s): 56242801  
- Comentarios: (empty text area)  
- Proveedor desde: (date picker input)  
- Dirección:

- Calle: Obraje
- Número exterior: 8
- Número interior: Nave 45-A
- País: (empty input)
- Estado: Edo. México
- Alcaldía/Municipio: Tlalnepantla
- CP: 54000
- Colonia/barrio/población: Industrial

  
At the bottom right are 'Imprimir' and 'Guardar' buttons.

Figura 94. Efecto de asignar el valor true a la etiqueta **showHintInField**.

## 5.- Añadir funcionalidad a los campos de un formulario

Como sucede en cualquier lenguaje de programación orientado al manejo de ventajas es posible en **SWBForms** el manejo de eventos que agregan funcionalidad o un comportamiento especializado a un campo. A continuación, se explican el proceso de manejo de eventos en **SWBForms**.

### 5.1.- Manejadores de eventos

Supongamos que tenemos un formulario como el que se muestra en la figura 88 para el manejo de la información sobre los proveedores de una tienda de conveniencia y se desea agregar la funcionalidad de que cuando suceda el evento de oprimir dos veces el botón derecho sobre uno de los proveedores que se muestran dentro de las filas de la tabla de la figura 95 se muestre un formulario que permita la actualización de dicha información como se muestra en la figura 96. Para lograr implementar esta funcionalidad deseada se debe hacer uso del manejo de eventos que proporcionan **SWBForms**.

	Nombre comercial	RFC	Correo electrónico	Sucursal	Teléfono(s)	
1	Semillas Nacionales SA SEN 551221AB1		contacto@senaisa.com	Tlalnepantla, EdoMex	56242801	✖
2	Bimbo	BIM 011108DJ5	contacto@gbimbo.com.mx	Santa Ma. Insurgentes	01 800 910 2030	✖
3	Coca-Cola	FEM 910826W25	--	Tlalpan	56242802	✖
4	Danone	DME 761202CP9	--	CDIS Vallejo	56242803	✖

Figura 95. Formulario con la lista de proveedores de la aplicación de control de una tienda de conveniencia que se desarrolla a lo largo de este manual.

The screenshot shows a software application window titled "iWise". The main title bar has the "iWise" logo and a "Terminar Sesión" button. The left sidebar menu includes "Inicio", "Compras" (Products, Suppliers, Purchase Orders), "Inventarios", "Ventas", and "Configuración". The main content area is titled "Proveedores" (Suppliers) and contains a form with the following fields:

Razón Social:	SeNaSA	Nombre comercial:	Semillas Nacionales SA de CV
RFC:	SEN 551221AB1	Correo electrónico:	contacto@senaisa.com
Sucursal:	Tlalnepantla, EdoMex	Página Web:	
Teléfono(s):	56242801	Comentarios:	
Proveedor desde:	<input type="text"/> <input type="button" value=""/>	Dirección	
Calle:	Obraje	Número exterior:	8
País:		Estado:	Edo. México
CP:	54000	Colonia/barrio/población:	Industrial

At the bottom right are "Imprimir" and "Guardar" buttons.

Figura 96. Formulario de actualización de la información de proveedores de una tienda de venta de productos de conveniencia

Para lograr el efecto antes descrito se debe de recurrir al manejo de eventos de **SWBForms**. En la figura 97 se muestra el código **SWBForms** para la implementación del manejo del evento deseado. En este ejemplo se implementa el evento **recordDoubleClick** en donde se indica que al oprimir en dos ocasiones sobre una fila de la tabla del formulario se ejecute un nuevo formulario para actualizar los valores de la fila seleccionada.

```
recordDoubleClick: function(grid, record) {
    window.location.href='/work/supplierMgr?suri='+record._id;
    return false;
},
```

Figura 97. Implementación del evento recordDoubleClick.

En la figura 98 se muestra el código completo del formulario sobre el que se implementa el evento recordDoubleClick.

```

<script type="text/javascript">
    var briefGrid;
    briefGrid = eng.createGrid({
        ID: "supplierDvr",
        title: "Proveedores",
        top: 10, width:"90%", height:"400", alternateRecordStyles:true, cellHeight:"32",
        titleOrientation: "top",
        autoFocus: true,
        showRollOverCanvas: false,
        showRollUnderCanvas: false,
        showRowNumbers: true,
        baseStyle: "boxedGridCellCyan",
        canAdd: true,
        canEdit: true,
        canRemove: true,
        showFilter: true,
        fields: [
            {name: "number", width: "20%"}, 
            {name: "supplier"}, 
            {name: "created"} 
        ],
        recordDoubleClick: function(grid, record) {
            window.location.href='/work/purchaseMgr?suri='+record._id;
            return false;
        },
        addButtonClick: function(event) {
            window.location.href = "/work/purchaseMgr";
            return false;
        }
    }, "PurchaseOrder");
</script>

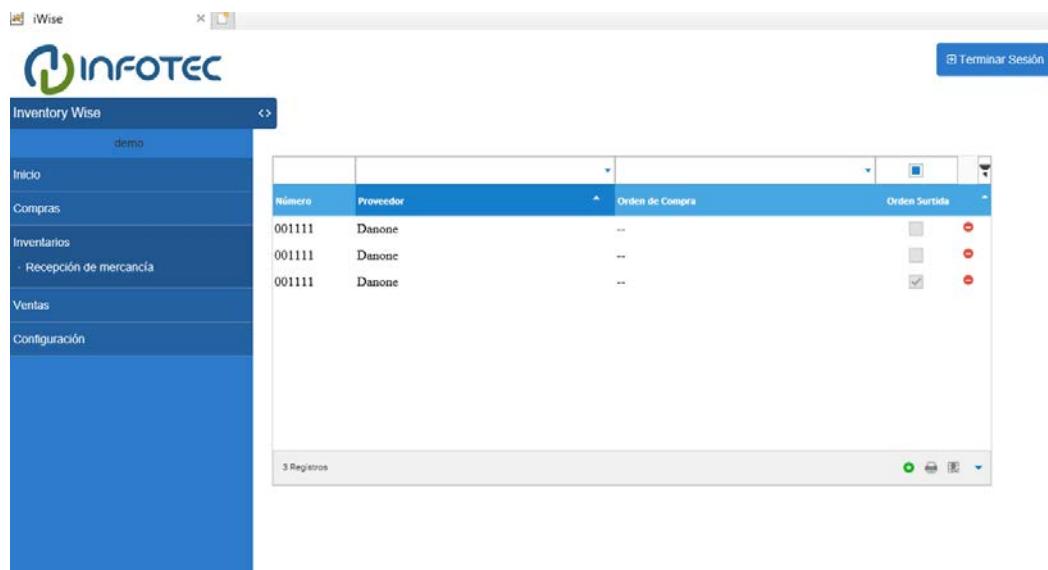
```

Figura 98. Código completo con la implementación del evento recordDoubleClick.

Como se puede observar en el código de la figura 98 la implementación de eventos en **SWBForms** se realiza dentro de la definición de formularios.

## 6.- Tablas

En esta sección se explican algunos aspectos relevantes respecto al manejo de tablas en **SWBForms**. Adicionalmente a lo comentado en la sección 3 un formulario también puede ser creado en una representación tabular como se muestra en la figura 99, mediante la etiqueta **eng.createGrid**.



The screenshot shows a software interface titled "Inventory Wise" with the logo "iWise" and "INFOTEC". The main menu on the left includes "Inicio", "Compras", "Inventarios" (with a sub-item "Recepción de mercancía"), "Ventas", and "Configuración". The central area displays a grid of data with the following columns: "Número" (Number), "Proveedor" (Supplier), "Orden de Compra" (Purchase Order), and "Orden Surtida" (Shipment Order). There are three rows of data, all from supplier "Danone" with "Número" 001111 and "Proveedor" Danone. The "Orden de Compra" column contains three entries: "..." (three dots). The "Orden Surtida" column contains three rows, each with a checkbox: the first two are unchecked (grey), and the third is checked (blue with a checkmark). At the bottom of the grid, it says "3 Registros".

Número	Proveedor	Orden de Compra	Orden Surtida
001111	Danone	...	
001111	Danone	...	
001111	Danone	...	

Figura 99. Representación tabular de un formulario en **SWBForms**.

### 6.1.- Filtros

Como se puede observar en la figura 100 la representación tabular del formulario incluye una sección de filtros automatizados que se resaltan en la figura 100. Estos filtros facilitan la búsqueda de información dentro de un formulario en representación tabular. Para implementar los filtros en un formulario dentro de SWBForms se debe de utilizar la etiqueta **showfilter** que permite activar el uso de filtros en una representación tabular de un formulario. En la figura 101 se muestra la visualización de un filtro en operación para facilitar la búsqueda de información dentro de un formulario en representación tabular.

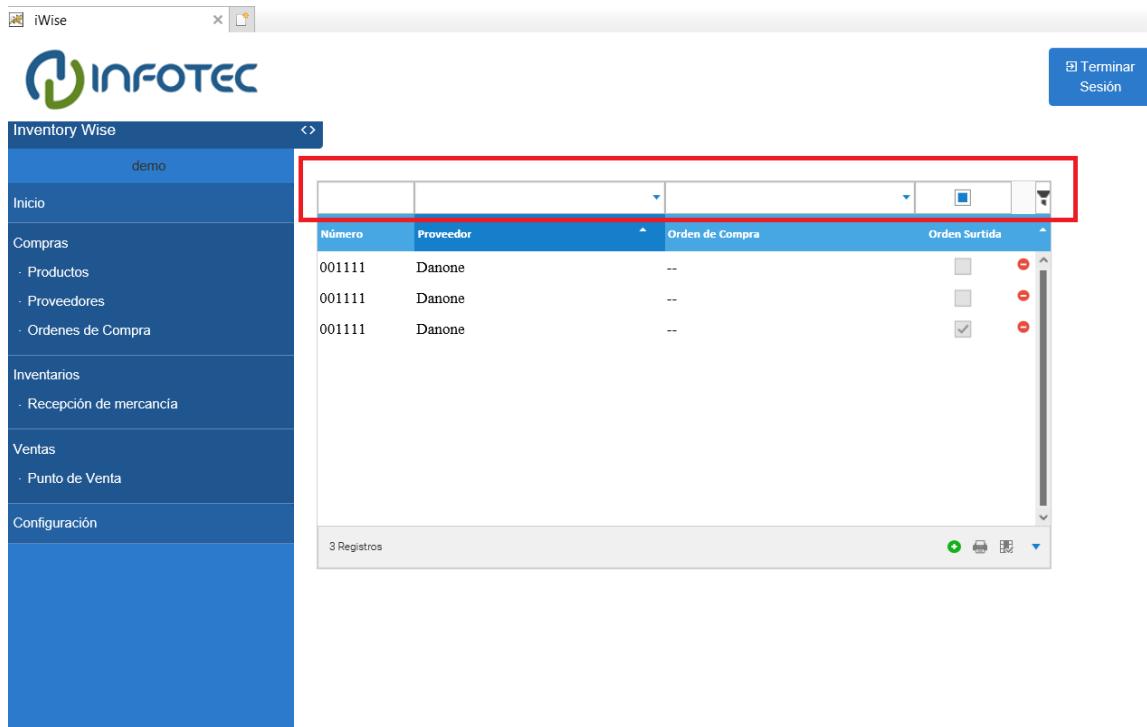


Figura 100. Formato tabular de un formulario con su sección de filtros activada (indicada en rojo dentro de la figura).

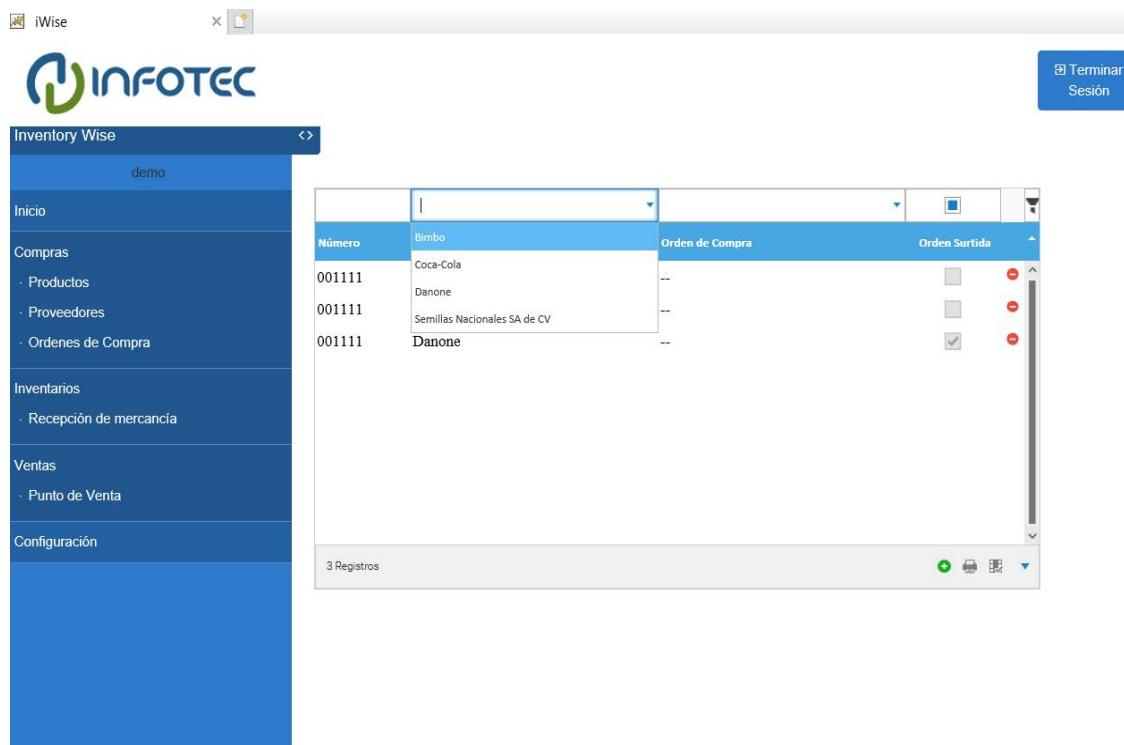


Figura 101. Filtro de un formulario en su representación tabular en acción.

En la figura 102 se muestra el código completo que permite implementar el formulario tabular que se muestra en las figuras 99,100 y 101.

```

<script type="text/javascript">
    var merRcvGrid;
    merRcvGrid = eng.createGrid({
        ID: "merRcvDvr",
        title: "Mis campañas",
        top: 10, width:"90%", height:"400", alternateRecordStyles:true, cellHeight:"32",
        titleOrientation: "top",
        autoFocus: true,
        showRollOverCanvas: false,
        showRollUnderCanvas: false,
        baseStyle: "boxedGridCellCyan",
        //editEvent: "click",
        //editByCell: false,
        canAdd: true,
        canEdit: true,
        canRemove: true,
        showFilter: true,
        sortDirection: "ascending",
        sortFieldNum: 1,
        fields: [
            {name: "number", width: 100},
            {name: "supplier"},
            {name: "purchaseOrder", stype: "select", dataSource: "PurchaseOrder"},
            {name: "status", width: 100, canEdit: false}
        ],
        recordDoubleClick: function(grid, record) {
            window.location.href='/work/merchandiseRcvMgr?suri=' + record._id;
            return false;
        }
    }, "MerchandiseRcv");
</script>

```

Figura 102. Código para la creación de un formulario tabular con la sección de filtros activada (etiqueta **showFilter: true**)

## 6.2.- GridView

La idea del GridView es la de un Grid, como los que se han visto a lo largo de este tutorial, con la característica que no permite realizar funciones de escritura sino únicamente de lectura, es decir, no es posible agregar, eliminar o modificar un registro del dataSource. Además, la información que se muestra en un GridView puede ser heterogénea, puede provenir de varios data sources. Para obtener la lista de datos, realizamos consultas con la ayuda de **SWB DataManager**.

Regularmente, y debido a su naturaleza, los GridView, se manejan como parte del elemento **links**. En la figura 103 y 104 se muestra un ejemplo del uso de la etiqueta GridView:

```

links: [
{
name: "membersRating",
    title:"Calificaciones",
    stype:"tab",
    fields: [
        {
            name: "rating",
            title:"Resumen de calificaciones",
            stype:"gridView",
            width:"100%", height: 400,
            alternateRecordStyles:true, cellHeight:"32",
            titleOrientation: "top",
            colSpan:6,
            baseStyle: "boxedGridCellRose",
            showRollOverCanvas:false,
            showRollUnderCanvas:false,
            fields: [
{name: "number", title:"Folio", type:"string"}
{name: "name", title:"Nombre del solicitante", type:"string"},
{name: "sg", title:"Grupo solidario", type:"string"},
{name: "paymentTimeRating", title:"Tiempo de pago", type:"int", width:120},
{name: "paymentRating", title:"Forma de pago", type:"int", width:100},
{name: "ratingTotal", title:"Calificación", type:"int", width:80},
{name: "loanAmountAppraisal", title:"Financiamiento Autorizado", type:"double", format: "\u0024,
0.00"}
                ],
            data:[<%
                DataList lst = gpReq.getDataList("accredited");
                if(lst!=null)
                {
                    Iterator<String> accredited = lst.iterator(); //iterador de acreditados (PLRs)
                    String plrId;
                    DataObject query, rs;
                    DataObject plr, prsn, bizPlan, payPlan;
                    int paymentTimeRating, paymentRating, paymentRatingTotal;

```

Figura 103. Ejemplo del uso de la etiqueta GridView (Parte 1)

```

        while(accredited.hasNext())
    {
        plrId = accredited.next();
        plr = plrDS.fetchObjById(plrId);
        if(plr==null)
            continue;
        query = new DataObject();
        query.addSubObject("data").addParam("loanRequest", plrId);
        rs = loanAppraisalDS.fetch(query);
        if(rs.getDataObject("response").getInt("totalRows")!=1)
            continue;
        prsn = personDS.fetchObjById(plr.getString("person"));
        if(prsn==null)
            continue;
        bizPlan = bizPlanDS.fetchObjById(plr.getString("businessPlan"));
        if(bizPlan==null)
            continue;
        payPlan = payPlanDS.fetchObjById(plr.getString("paymentPlan"));
        if(payPlan==null)
            continue;
        out.print("{");
        out.print("number:"+"'"+plr.getString("folio")+"'");
        out.print(", name:"+"'"+prsn.getString("firstName")+" "+prsn.getString("lastName")+""
"+prsn.getString("secLastName","","")+"'");
        out.print(", sg:"+"'"+solGrp.getString("groupName")+"'");
        paymentTimeRating = (int)payPlan.getDouble("paymentTimeRating");
        paymentRating = payPlan.getInt("paymentRating");
        paymentRatingTotal = paymentTimeRating + paymentRating;
        out.print(", paymentTimeRating:"+paymentTimeRating+","
paymentRating:"+paymentRating);
        out.print(", ratingTotal:"+paymentRatingTotal);
        out.print(", "
loanAmountAppraisal:"+rs.getDataObject("response").getDataList("data").getDataObject(0).getDouble(
"loanAmountAppraisal"));
        out.print("}");
        if(accredited.hasNext()) {
            out.println(",");
        }else {
            out.println("");
        }
    }
}
%>
        ] // data
    } // rating
] // membersRating
}

] // links

```

Figura 104. Ejemplo del uso de la etiqueta GridView (Parte 2)

### 6.3.- Implementar catálogos

Un uso muy importante de la representación tabular de un formulario es la facilidad para la creación de un catálogo de fuentes de datos que pueda fácilmente ser visualizado en una página web. En la figura 105 se muestra un código ejemplo para la creación de un catálogo de la información los productos de una tienda de conveniencia.

```
<h3>Catálogo de Productos</h3>
<script type="text/javascript">
    var grid = eng.createGrid({
        width: "95%", height: 400, alternateRecordStyles:true, cellHeight:"30",
        showRolloverCanvas: false,
        showRollUnderCanvas: false,
        showRowNumbers: false,
        showFilter: true,
        wrapCells: true,
        editEvent: "click",
        editByCell: true,
        canReorderFields: true,
        canResizeFields: true,
        canEdit: true,
        canRemove: true,
        canAdd: true,
        sortDirection: "ascending",
        sortFieldNum: 1,
        rowContextClick: "isc.say(record._id); return false;",
        fields:[
            {name: "code"}, 
            {name: "productName"}, 
            {name: "description"}, 
            {name: "CFDI", width: 150, align: "center"}, 
            {name: "CFDIU", width: 150, align: "center"}, 
            {name: "updated"}, 
            {name: "created"} 
        ]
    }, "Product");
</script>
```

Figura 105. Código ejemplo para la creación de un catálogo de una fuente de datos en **SWBForms**.

Como resultado de la ejecución del código de la figura 105 se visualiza el catálogo que se muestra en la figura 106. Esta representación facilita la visualización, adición y edición de los datos almacenados en las fuentes de datos de la aplicación que se esté desarrollando.

## Catálogo de Productos

Código	Nombre	Descripción	Clave CFDI	Unidad CFDI	Fecha de actualiz...	Fecha de creación	
2052	Aqua Bonafont 1.5 L	Aqua embotellada Bonafont 1.5 L	Bebidas no alcohólicas	Pieza	--	--	
2050	Aqua Bonafont 355 ml.	Aqua embotellada Bonafont 355 ml.	Bebidas no alcohólicas	Pieza	--	--	
2051	Aqua Bonafont 600 ml.	Aqua embotellada Bonafont 600 ml.	Bebidas no alcohólicas	Pieza	--	--	

Figura 106. Ejemplo de visualización de un catálogo mediante un formulario tabular en **SWBForms**.

## 7.- Configuración del esquema de usuarios y permisos de acceso

**SWBForms** proporciona en toda aplicación de forma pre-instalada dos scripts que para el registro de usuarios de la aplicación y el control de acceso a la aplicación que se está desarrollando de forma tal que una aplicación en **SWBForms** ya incluye un esquema seguridad en toda aplicación y sin necesidad de realizar codificación alguna. Estos scripts se ubican en la ruta work/config/ y tienen el nombre login.jsp y register.jsp. El primero de estos scripts sirve para validar si un usuario se ha registrado en la aplicación que se está desarrollando y el segundo permite dar de alta un nuevo usuario en la aplicación. En la figura 107 se muestra parte del código del script login.jsp. Como se puede observar mediante el manejo de objetos y de la función fetch() se puede fácilmente validar el acceso a usuarios de la aplicación que se está desarrollando.

```

<%@page import="org.semanticwb.datamanager.*"%><%
    String email=request.getParameter("email");
    String password=request.getParameter("password");
//System.out.println(email+" "+password);
if(email!=null && password!=null)
{
    SWBScriptEngine engine=DataMgr.initPlatform(session);
    SWBDataSource ds=engine.getDataSource("User");
    DataObject r=new DataObject();
    DataObject data=new DataObject();
    r.put("data", data);
    data.put("email", email);
    data.put("password", password);
    DataObject ret=ds.fetch(r);
//engine.close();

    DataList rdata=ret.getDataObject("response").getDataList("data");

    //System.out.println("ret"+ret);
    if(!rdata.isEmpty())
    {
        session.setAttribute("_USER_", rdata.get(0));
        response.sendRedirect("/");
        return;
    }
}
%>
```

Figura 107. Script dentro del archivo login.jsp para el control de acceso a una aplicación web desarrollada mediante **SWBForms**.

Ahora bien para el registro de un usuario en una aplicación que se está desarrollando mediante **SWBForms** se proporciona un script modelo en el archivo register.jsp, ver figura 108. Como puede observarse mediante el método addObj es fácil almacenar los datos de un nuevo usuario de la aplicación que se está desarrollando en la fuente de datos User.

```

<%@page contentType="text/html" pageEncoding="UTF-8"%>
<%@page import="org.semanticwb.datamanager.*"%><%
    String fullname=request.getParameter("fullname");
    String email=request.getParameter("email");
    String password=request.getParameter("password");
    String password2=request.getParameter("password2");
    if(email!=null && password!=null)
    {
        if(password.equals(password2))
        {
            SWBScriptEngine engine=DataMgr.initPlatform(session);
            SWBDataSource ds=engine.getDataSource("User");
            DataObject obj=new DataObject();
            obj.put("fullname", fullname);
            obj.put("email", email);
            obj.put("password", password);
            ds.addObj(obj);
//engine.close();
            response.sendRedirect("/login");
            return;
        }
    }
%>
```

Figura 108. Script dentro del archivo register.jsp para el registro de usuarios en una aplicación que se está desarrollando mediante SWBForms.

Adicionalmente, para realizar el control de usuarios y de sesión de los mismos es necesario como primer paso incluir en la página principal de la aplicación que se está desarrollando el código **SWBForms** mostrado en la figura 109.

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<%@page import="org.semanticwb.datamanager.*"%>
<%
    SWBScriptEngine engine = DataMgr.getUserScriptEngine("/work/jsp/datasources.js", (DataObject)null, false);
    String login = null;
    String fullname = null;
    DataObject user = null;

    if(session.getAttribute("_USER_") != null) {
        user = (DataObject)session.getAttribute("_USER_");
        if(user != null) {
            if(user.getClassName().equals("User")) {
                login = user.getString("login", null);
                fullname = user.getString("fullname");
            }
        }
    }
%>
```

Figura 109. Script ejemplo para el control de la sesión y control de acceso a usuarios de una aplicación que se está desarrollando mediante **SWBForms**.

Finalmente, para realizar concluir con el control de sesión de usuarios en la aplicación que se está desarrollando basta con que se incluya al principio de cada uno de los archivos .jsp de la aplicación, el código de la figura 110.

```
final DataObject user = (DataObject) session.getAttribute("_USER_");
```

Figura 110. Código **SWBForms** que permite el control de sesión en una aplicación Web.

Esta capacidad que tiene **SWBForms** de llevar el control de acceso a la aplicación y sesión de usuarios simplifica mucho el desarrollo de aplicaciones Web.

## 7.1.- Usuario, Roles y Grupos de usuarios

Adicionalmente al control de acceso a usuarios y seguimiento de sesiones de usuarios, en muchas aplicaciones es deseable el control de acceso a funciones de la aplicación que se está desarrollando mediante la definición de roles y grupos de usuarios. En la aplicación ejemplo que se desarrolló para ejemplificar los conceptos básicos de **SWBForms** se han desarrollado formularios para el manejo de usuarios, roles y grupos. En la figura 111 se muestra el formulario para que permite el alta y edición de usuarios y la asignación de roles y grupos.

Figura 111. Formulario de alta de usuarios y asignación de roles y grupos.

El formulario mostrado en la figura 111 puede ser creado fácilmente en **SWBForms**, en la figura 112 se muestra el código para crear este formulario.

```
<script type="text/javascript">
    var usrId = '<%=suri%>';
    var usrFrm;
    usrFrm = eng.createForm({
        ID:"usrMngr"
        ,title:"Usuario"
        ,top: 10, width: "95%", height: "*"
        ,numCols: 4
        ,titleOrientation: "left"
        ,showTabs: true
        ,autoFocus: true
        ,baseStyle: "boxedGridCellCyan"
        ,canAdd: true
        ,canEdit: true
        ,canRemove: true
        //,titleSuffix: ":"
        //,requiredTitleSuffix: "<span style='color:#DE3163'></span></B>"
        ,fields: [
            {name: "login"}, 
            {name: "email"}, 
            {name: "roles", colSpan: 3, width: "100%"}, 
            {name: "groups", colSpan: 3, width: "100%"}]
        ,links: [
            {name: "person", title: "Datos de Perfil", type: "Person", stype: "subForm", dataSource: "Person"}]
    }, usrId, "User");
</script>
```

Figura 112. Código **SWBForms** para crear el formulario de la figura 111.

De igual manera para crear grupos de perfiles de usuarios se puede crear un formulario en formato tabular fácilmente en **SWBForms** como el que se muestra en la figura 113.

### Grupo de Permisos de Usuario

Perfil	
1	Punto de Venta
2	Compras
3	Configuración
4	Inventarios
4 Registros	

Figura 113. Formulario de alta y edición de grupos

Para crear el formulario de la figura 113 se requiere el código de la figura 114. Como puede observarse se facilita el control de usuarios, roles y grupos en **SWBForms**.

```
<h3>Grupo de Permisos de Usuario</h3>
<script type="text/javascript">
    var profileGrid;
    profileGrid = eng.createGrid({
        ID: "profileDvr",
        title: "Perfiles de usuario",
        top: 10, width:"90%", height:"250", alternateRecordStyles:true, cellHeight:"32",
        titleOrientation: "top",
        autoFocus: true,
        showRollOverCanvas: false,
        showRollUnderCanvas: false,
        showRowNumbers: true,
        baseStyle: "boxedGridCellCyan",
        //editEvent: "click",
        //editByCell: false,
        canAdd: true,
        canEdit: true,
        canRemove: true,
        showFilter: false,
        //sortDirection: "descending",
        //sortFieldNum: 3,ssss
    }, "Profile");
</script>
```

Figura 114. Código **SWBForms** para crear el formulario de la figura 113.

## 7.2.- Gestionar usuarios y permisos

Finalmente, en la figura 115 se muestra un formulario ejemplo que facilita la asignación de permisos a grupos de usuarios.

Permisos de Usuario				
	Perfil	Nombre	Descripción	Roles
25	Compras	viewListPurchaseOrder	Permiso para ver lista de órdenes de compra	Órdenes de Compra
26	Compras	viewPurchaseOrder	Permiso para ver detalle de orden de compra	Órdenes de Compra
27	Compras	addSupplier	Permiso para agregar nuevo proveedor	Proveedores
28	Compras	delSupplier	Permiso para borrar proveedor	Proveedores
29	Compras	editSupplier	Permiso para modificar proveedor	Proveedores
30	Compras	viewListSupplier	Permiso para ver lista de proveedores	Proveedores
31	Compras	viewSupplier	Permiso para ver detalle de proveedor	Proveedores
32	Configuración	addUser	Permiso para agregar nuevo usuario	Gerencia
33	Configuración	delUser	Permiso para borrar usuario	Gerencia
34	Configuración	editUser	Permiso para modificar usuario	Gerencia
35	Configuración	viewListUser	Permiso para ver lista de usuarios	Gerencia
36	Configuración	editCatalog	Permiso para modificar información de catálogo	Gerencia
37	Configuración	delCatalog	Permiso para borrar información de catálogo	Gerencia
38	Configuración	viewCatalog	Permiso para ver catálogos	Gerencia
39	Configuración	addPermission	Permiso para agregar nuevo permiso	Gerencia
40	Configuración	delPermission	Permiso para borrar permiso	Gerencia
41	Configuración	grantPermission	Permiso para otorgar permiso a usuario	Gerencia
42	Punto de Venta	openPointSale	Permiso para abrir punto de venta	Ventas Supervisor
43	Punto de Venta	settingPointSale	Permiso para configurar punto de venta	Ventas Supervisor
44	Punto de Venta	registerSale	Permiso para registrar venta	Ventas Servidor

Figura 115. Formulario para el manejo de permisos en una aplicación mediante **SWBForms**.

El formulario de la figura 115 se puede crear de forma muy sencilla en **SWBForms** mediante el código de la figura 116.

```
<h3>Permisos de Usuario</h3>
<script type="text/javascript">
var permissionGrid;
permissionGrid = eng.createGrid({
    ID: "permissionDvr",
    title: "Permisos de usuario",
    top: 0, width:"90%", height:"600", alternateRecordStyles:true, cellHeight:"32",
    titleOrientation: "top",
    autoFocus: true,
    showRollOverCanvas: false,
    showRollUnderCanvas: false,
    showRowNumbers: true,
    baseStyle: "boxedGridCellCyan",
    cellHeight: 24,
    //editEvent: "click",
    //editByCell: false,
    canAdd: true,
    canEdit: true,
    canRemove: true,
    showFilter: true,
    //sortDirection: "ascending",
    //sortFieldNum: 0,
    fields:[
        {name:"profile", width: 150},
        {name:"name", width: 150},
        {name:"description"},
        {name:"roles", width: 200}
    ]
}, "Permission");
</script>
```

Figura 116. Código **SWBForms** para crear el formulario de la figura 115.

## 8.- DataProcessors

Al igual que otros entornos de programación para aplicaciones web en **SWBForms** existe la posibilidad de realizar procesamiento automatizado de los datos de las fuentes de datos desde los scripts de **SWBForms**. Una forma de realizar esto es mediante los llamados **dataProcessors**. En la figura 117, se muestra un **dataProcessor**, una porción de código que realiza una función específica, sobre una fuente de datos. En el ejemplo se modifican los campos *create* y *update* de la fuente de datos *PurchaseOrder* (Orden de compra) con la fecha actual siempre que el usuario quiera agregar o actualizar un nueva orden de compra. Este monitoreo de las acciones del usuario se realiza de forma automática y el código del *dataProcessor* se ejecuta de forma autónoma cuando se detectan las acciones de adición y/o actualización sobre la fuente de datos.

```
eng.dataProcessors["dpPurchaseOrder"] = {  
    dataSources: ["PurchaseOrder"],  
    actions: ["add", "update"],  
    request: function(request, dataSource, action)  
    {  
        print("\n\n dpPurchaseOrder.....");  
        var today = getFormatDate();  
        if(action==="add")  
        {  
            request.data.created = today;  
            request.data.updated = today;  
        }  
        else if(action==="update")  
        {  
            request.data.updated = today;  
        }  
        return request;  
    },  
    response: function (response, dataSource, action)  
    {  
        return response;  
    }  
};
```

Figura 117. Ejemplo de la implementación de un **dataProcessors**,

Figura 118. Efecto del uso de un **dataProcessors** sobre el campo fecha de creación en el formulario órdenes de compra de la aplicación de venta de productos de conveniencia.

Como se puede observar en la figura 118 el campo fecha de creación no es editable ya que se llena de forma automática con la fecha en la cual se actualiza o da de alta una orden de compra. Esto se logra gracias al **dataProcessors** de la figura 117.

## 9.- DataServices

Otra forma alternativa de realizar el procesamiento de fuentes de datos de forma automática es el **dataServices** al igual que el **dataProcessors** realiza el procesamiento de los datos de una fuente de datos de forma automática. En el ejemplo de la figura 119 se implementa un **dataServices** que permite actualizar las existencias en el inventario de productos una vez que se ha surtido una orden de compra en la aplicación de venta de productos de una tienda de conveniencia.

```

eng.dataServices["dsMerchandiseRcv"] = {
    dataSources: ["MerchandiseRcv"],
    actions: ["update"],
    service: function (request, response, dataSource, action) {
        var productDS = this.getDataSource("Product");
        var itemOrderDS = this.getDataSource("ItemOrder");

        var prdDO;
        var itemOrderDO;
        if(request.data.status==true && request.oldValues.status==false) {
            for(var i=0; i<request.data.order.size(); i++) {
                itemOrderDO = itemOrderDS.fetchObjById(request.data.order[i]);
                if(itemOrderDO) {
                    prdDO = productDS.fetchObjById(itemOrderDO.product);
                    if(prdDO) {
                        if(itemOrderDO.quantity>0) {
                            prdDO.existence += itemOrderDO.quantity;
                            productDS.updateObj(prdDO);
                        }
                    }
                }
            }
        }
    };
};

```

Figura 119. Ejemplo de la implementación de un **dataServices**.

El dataServices de la figura 119 es utilizada de forma asociada al formulario de 120 para actualizar las existencias de productos en la aplicación desarrollada en el presente manual.

Unidad	Producto	Cantidad	Costo	Iva	Importe
Paquete	Aqua Bonafont 1.5 L	1	\$1.52	0%	\$1.52
Paquete	Aqua Bonafont 355 ml.	1	\$2.65	0%	\$2.65
Paquete	Aqua Bonafont 600 ml.	1	\$5.96	0%	\$5.96

Figura 120. Formato de recepción de mercancía en donde se resalta la casilla orden surtida.

Finalmente, en la figura 121 se ve el efecto del dataServices de la figura 119 al actualizar las existencias de productos de una tienda de conveniencia.

The screenshot shows a web-based application titled "Inventory Wise" with the "demo" environment selected. The main menu on the left includes "Inicio", "Compras" (Products, Suppliers, Purchase Orders), "Inventarios" (Reception of goods), "Ventas", and "Configuración". The central area displays a table of products with columns: Código (Code), Nombre (Name), Descripción (Description), Clave CFDI (CFDI Key), and Existencia (Stock). The table shows three entries for Bonafont water: 2052 (1.5 L), 2050 (355 ml), and 2051 (600 ml). All three rows have a red circle icon next to their stock counts (1, 5, and 4 respectively), indicating they are out of stock. A message at the bottom of the table says "3 Registros".

Código	Nombre	Descripción	Clave CFDI	Existencia
2052	Aqua Bonafont 1.5 L	Aqua embotellada Bonafont 1.5 Bebidas no alcohólicas		1 <span style="color:red;">●</span>
2050	Aqua Bonafont 355 ml.	Aqua embotellada Bonafont 355Bebidas no alcohólicas		5 <span style="color:red;">●</span>
2051	Aqua Bonafont 600 ml.	Aqua embotellada Bonafont 600Bebidas no alcohólicas		4 <span style="color:red;">●</span>

Figura 121. Efecto de la aplicación del dataServices de la figura 119 sobre la actualización de las existencias en los productos de la tienda de conveniencia.

## 10.- Implementar bitácoras de seguridad

Las bitácoras de seguridad registran las diferentes actividades o acciones que realiza cada usuario dentro del Sistema. Cabe mencionar que, para identificar las acciones de un usuario específico, es necesario que dicho usuario se encuentre en una sesión activa de la aplicación, de lo contrario, la información registrada solo hará referencia a los objetos y clases afectados en cada acción que realizó el usuario.

Para implementar una bitácora de seguridad, sobre las acciones que realizan los distintos usuarios registrados de una aplicación, lo primero, es definir una fuente de datos para almacenar los datos de las actividades realizadas dentro del sistema, ver figura 122.

```

/**
 * Log
 * Bitácora de acciones realizadas por los diferentes usuarios
 * @type dataSource
 */
eng.dataSources["Log"] = {
    scls: "Log",
    modelid: "CONACyT",
    dataStore: "mongodb",
    displayField: "user",
    fields: [
        {name: "source", title: "Source", type: "string"},
        {name: "usr", title: "Usuario", type: "string"},
        {name: "usrIp", title: "IP Usuario", type: "string"},
        {name: "dataSource", title: "DataSource", type: "string"},
        {name: "action", title: "Acción", type: "string"},
        {name: "data", title: "Datos", type: "boolean"}, //Se define como boolean para que
internamente se interprete como objeto
        {name: "timestamp", title: "TimeStamp", type: "long"},
    ],
};

```

Figura 122. Código para la creación de la fuente de datos para implementar una bitácora de seguridad.

Posteriormente, definimos el dataServices que realiza el registro en Base de Datos de los datos de cada una de las acciones realizadas en la aplicación, ver figura 123.

```

/** LogsService
 * Servicio que registra el log de la bitácora de las diferentes acciones realizadas
 * dentro de la aplicación
 * @type dataService */
eng.dataServices["LogsService"] = {
    dataSources: ["*"],
    actions: ["add", "remove", "update"],
    service: function (request, response, dataSource, action)
    {
        if (dataSource !== "Log")
        {
            var data = {
                source: this.source,
                usr: "",
                usrIp: "",
                dataSource: dataSource,
                action: action,
                data: [request.data],
                timestamp: Date.now(),
            };
            if (this.user)
            {
                data.usr = this.user.email;
                data.usrIp = this.user.IP;
            }
            //print("saveLog:===="+data);
            this.getDataSource("Log").addObj(data);
        }
    }
};

```

Figura 123. Código del dataServices para implementar una bitácora de seguridad.

Este Data Service no registra las acciones referentes a lecturas o vistas, pero SWB Forms también permite registrarlas, para tal efecto, basta con agregar al arreglo de acciones la palabra reservada “fetch”.