IVAP 2017. Formación especializada. Curso 2017/ES/1646/27/46/1

Análisis, diseño e implementación de aplicaciones con el framework corporativo gvHIDRA

David Pascual Serral

Jorge Belenguer Faguás

Verónica Navarro Porter

Índice de contenido

1 Introducción

La intención del curso presente es la de introducir al alumno a la tecnología gvHidra, aunque tratándolo desde un punto de vista algo distinto respecto a ediciones anteriores. Los alumnos del presente curso realizan (principalmente) tareas de supervisión, análisis, estimaciones y gestión de proyectos, y la programación tiene un menor peso en el día a día (aunque no se deja de lado). La mayoría de los alumnos son responsables (o pueden serlo) de aplicaciones realizadas con esta tecnología.

Los objetivos del curso son por tanto:

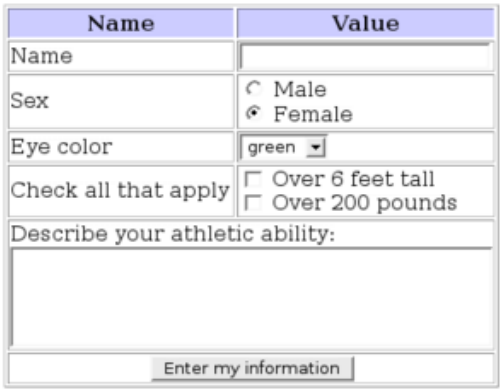
- Conocer la terminología habitual en el desarrollo Web y comprender su particularidades.
- Conocer la tecnología gvHidra, sus virtudes, puntos fuertes y sus límites o defectos.
- Tener suficientes nociones de arquitectura y programación de aplicaciones gvHidra para localizar la fuente de un error.
- Realizar estimaciones sobre el esfuerzo de desarrollo con esta tecnología así como supervisar estimaciones realizadas por otros.
- Conocer aspectos básicos de la programación de aplicaciones en el FW gvHidra.

Estos objetivos se enmarcan en la nueva forma de funcionamiento de la DGTI que basa el desarrollo y mantenimiento de sus aplicativos SW en un modelo de gestión orientado a servicio, cuyo principal representante sería el contrato DESIG.

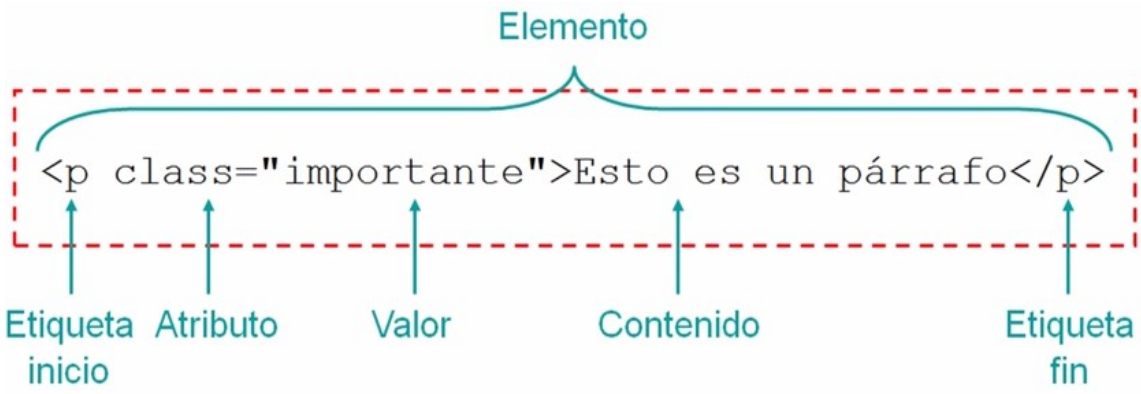
1.1 El desarrollo Web, conceptos básicos

- HTML (Lenguaje de marcado de hipertexto). Lenguaje de marcado estándar (W3C) para la elaboración de páginas web, se inicia en 1989 y actualmente se trabaja en su la versión HTML5. A lo largo de su evolución ha ido incorporando al estándar mejoras que le han permitido convertirse en un plataforma de programación.

- FORM: Los formularios Web permiten que el usuario interaccione con el navegador introduciendo información que será recibida posteriormente en el servidor Web vía GET y POST (aunque los motores JavaScript actuales permiten una vía asíncrona mediante *XMLHttpRequesty*). Es el principal responsable de que la plataforma Web se haya convertido en una plataforma de desarrollo de aplicaciones.



- CSS: Hoja de estilo en cascada o CSS (siglas en inglés de *cascading style sheets*). Lenguaje usado para definir la presentación de un documento estructurado escrito en HTML. En la evolución de HTML se ha tratado de separar la semántica del lenguaje de la presentación (que se ha cedido a otros lenguajes cuyo principal representante es CSS y estándar de facto). Define la presentación de los documentos HTML. Por ejemplo, CSS abarca cuestiones relativas a fuentes, colores, márgenes, líneas, altura, anchura, imágenes de fondo, posicionamiento. Luego veremos un apartado concreto sobre CSS, diseños responsive y Bootstrap.



- JavaScript (JS): Lenguaje de programación interpretado que se **ejecuta en el lado del cliente** (en el navegador Web). No tiene nada que ver con el Java (estrategia de Netscape). Ha sido un lenguaje de juguete hasta que las plataformas móviles lo han encumbrado (y evolucionado) siendo hoy un protagonista importantísimo (sino el principal) en el actual desarrollo de aplicaciones (web/móviles).

- Servidor Web: Ejecutado como indica su nombre en el lado del servidor, se comunica con el lado cliente (navegador web) a través del protocolo estándar HTTP. Apache ha sido el servidor HTTP que ha dominado con rotundidad (aún lo hace) el mundo Web, sin embargo, actualmente hay multitud de alternativas, tanto propietarias, como basadas en *opensource*. El funcionamiento básico de un servidor Web responder a peticiones de contenido, dicho contenido puede ser **estático**, o **dinámico**.
  - Espera peticiones HTTP (TCP/IP puerto 80).
  - Recibe una petición.
  - Busca el recurso (si es estático lo gestiona el mismo, y si es dinámico entra en juego otro elemento).
  - Envía el recurso utilizando la misma conexión por la que recibió la petición.
  - Vuelta a esperar.

- Navegador Web: Aplicación SW que permite acceder a Internet, y navegar introduciendo una URL. Contiene normalmente un motor JavaScript. Actualmente los navegadores más utilizados son (con diferencia) Google Chrome y Mozilla Firefox (fuente: http://www.w3schools.com/browsers/browsers_stats.asp). Internet explorer ha perdido la batalla, aunque a partir de la versión IE10 ha empezado a replantearse el cumplimiento de estándares debido al abandono por parte de los usuarios del navegador.

- Lenguajes de programación Web (servidor): Se ejecuta en el lado del servidor web (que cede la información vía CGI o módulo al motor del lenguaje). El lenguaje genera la página que va a servirse al navegador total o parcialmente y se envía. Las páginas que se ejecutan en el servidor pueden realizar accesos a bases de datos, conexiones en red, y otras tareas para crear la página final que verá el cliente. Actualmente, existen multitud de lenguajes diseñador para trabajar en el desarrollo web en el lado del servidor, pero podemos destacar (en orden alfabético): ASP, Java, PHP, Python, Ruby.

- SGBD (Sistema de gestión de base de datos): Aplicativo SW que permite almacenar, actualizar, extraer y manipular información en una base de datos. Destacaremos los sistemas relacionales como los propietarios Oracle, MS SQL Server, Informix o los *opensource* como

PostgreSQL, mariaDB o SQLite. Actualmente el mundo web, en proyectos de mucho volumen (redes sociales, buscadores) está haciendo uso de SGBDs noSQL, como MongoDB, Apache Cassandra, BigTable...

El siguiente apartado abordaremos aspectos básicos de la programación con lenguajes Web y particularidades concretas del lenguaje de programación PHP.

Como previo, en la figura siguiente se recoge una visión global de como interactúan los elementos que acabamos de definir con el PHP como lenguaje de programación en el lado del servidor:

1.1.1 CSS, diseño *responsive* y bootstrap

Ya hemos comentado brevemente que CSS se ha adoptado como el lenguaje encargado de dar formato al HTML. En los últimos tiempos, dado que se accede a Internet desde una gran cantidad de dispositivos móviles, tablets, ordenadores, etc.) se ha puesto de moda el término inglés "*responsive*" que se traduce al castellano como "diseño adaptativo".

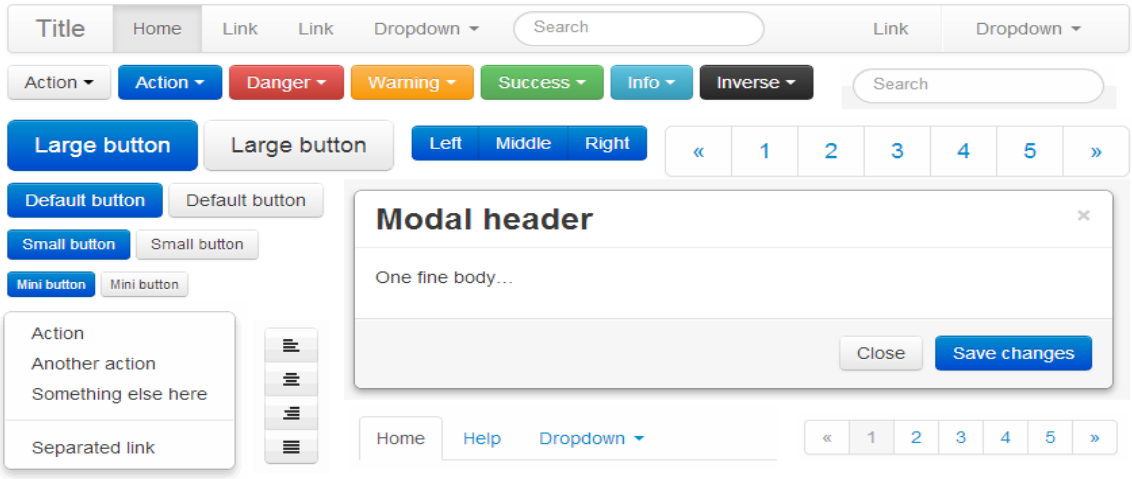
El diseño web *responsive* o adaptativo no es más que aplicar técnicas de diseño web en las CSS y HTML de forma que se consiga la correcta visualización de una misma página en distintos dispositivos. Es decir, redimensionar y colocar los elementos de la web de forma que se adapten al ancho de cada dispositivo permitiendo una correcta visualización y una mejor experiencia de usuario. El diseño *responsive* es una práctica casi obligatoria en el diseño Web actual.

Para conseguir de forma ágil diseños de este tipo, surgen distintos frameworks CSS que asisten en el diseño adaptativo, siendo bootstrap uno de los principales (si no el principal, aunque hay otros como Foundation, Skeleton, YAML, Pure CSS) framework o conjuntos de herramientas basado en software libre.

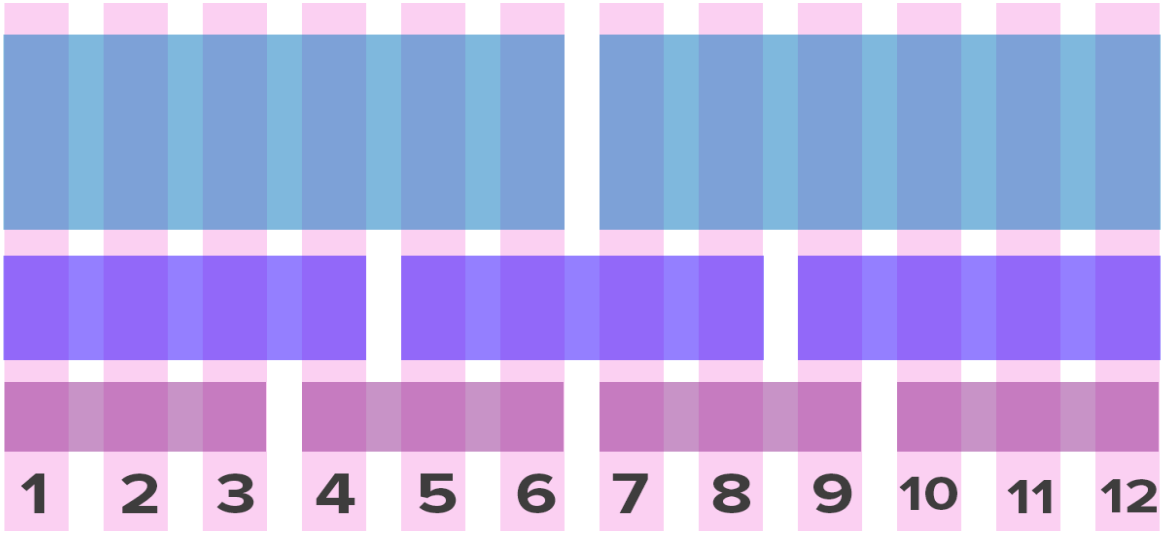
Bootstrap está desarrollado por Twitter por y publicado bajo la licencia *opensource* "Apache 2", donde destacan:

- Mantenimientos y actualizaciones realizados por Twitter, es un framework que mejora y evoluciona.
- Amplia comunidad que colabora e incorpora nuevos desarrollos (gráficos, imágenes, elementos de interfaz)
- Se basa en los lenguajes WEB aceptados como estándares de facto, HTML (HTML5), CSS (CSS3) Y JavaScript.
- Herramienta sencilla y ágil para crear sitios web e interfaces. Una vez que se entiende su funcionamiento, es fácil diseñar *interfaces* de forma rápida.

- Trae bastantes elementos por defecto, botones, mensajes de alerta, ventanas emergentes…



- Incluye un sistema de *grid* o rejilla, este sistema sirve para estructurar en la pantalla el contenido a visualizar. Por defecto incluye 12 columnas fijas (fluidas), dependiendo de si tu diseño será adaptativo o no.



- El número de columnas se va adaptando según sea la resolución de la pantalla del dispositivo

	Dispositivos muy pequeños Teléfonos (<768px)	**Dispositivos pequeños** Tablets (≥768px)	**Dispositivos medianos** Ordenadores (≥992px)	**Dispositivos grandes** Ordenadores (≥1200px)
Comportamiento	Las columnas se muestran siempre horizontalmente.	Si se estrecha el navegador, las columnas se muestran verticalmente. A medida que aumenta su anchura, la rejilla muestra su aspecto horizontal normal.		
Anchura máxima del contenedor	Ninguna (`auto`)	`728px`	`940px`	`1170px`
Prefijo de las clases CSS	`.col-xs-`	`.col-sm-`	`.col-md-`	`.col-lg-`

- Es compatible con la mayoría de navegadores web, y más desde la versión 3 (en la que se apoya gvHidra):
  - Google Chrome (en todas las plataformas)
  - Safari (tanto en iOS como en Mac)
  - Mozilla Firefox (en Mac y en Windows)
  - Internet Explorer 9.0 o superior (en Windows y Windows Phone)
  - Opera (en Windows y Mac).

Podemos encontrar más información en la Web de bootstrap. En la web vamos a encontrar 5 apartados:

- Getting Started: En este apartado explica como instalar Bootstrap en nuestra Web/aplicación.
- CSS: Aquí veremos los elementos HTML que invocan a los estilos CSS, una de las cosas más importantes de este apartado es el sistema de *grid* que ya hemos comentado.
- Components: Más de una docena de componentes reutilizables construidos para proporcionar la iconografía, menús desplegables, navegación, alertas… Un punto importante de este apartado son los Glyphicon. Los Glyphicon no son imágenes, son **fuentes** y por lo tanto si se desea modificar el color o tamaño se tendrá que utilizar los estilos CSS: ***font-size*** (tamaño del Glyphicon) y ***color*** (color del Glyphicon).
- JavaScript: Elementos realizados con jQuery. En este apartado podemos ver aquellos elementos que necesitan jQuery como son las ventanas emergentes, *slide*, menús...
- Customize: En el caso que queramos instalar Bootstrap pero solo nos interese un paquete en particular, podemos obtener nuestra instalación personalizada.

1.2 El lenguaje PHP

1.2.1 Historia

PHP es el acrónimo de PHP Hypertext Preprocessor[1], es un acrónimo recursivo. Es un lenguaje de programación del lado del servidor. Fue desarrollado en 1994 por el programador groenlandés Rasmus Lerdof. Las principales características que han hecho popular al lenguaje son:

- *Opensource*
- Multiplataforma
- Sencillo, versátil y rápido de aprender.
- Rápido
- Consume pocos recursos
- Gran cantidad de librerías y extensiones (PECL, PEAR, PHPClasses...)
- Gran comunidad
- Amplia y excelente documentación

En sus orígenes PHP es un lenguaje de programación estructurada, pero con la publicación de PHP 4.0 en 1998 comienza a incorporar características de la POO y es en 2004 con la publicación de PHP5 cuando se convierte de forma total en un lenguaje OO, por lo que las características del lenguaje a partir de su versión 5 se incrementan.

- POO (Clases, herencia, interfaces, traits...)
  - Nuevas clases que mejoran el lenguaje como DateTime
  - Escalabilidad
- Tratamiento de excepciones (try/catch)
- Uniformidad en la conexión a BD (PDO)
- Manejo nativo de XML: SOAP, XSLT, XPATH

El lenguaje PHP es utilizado tanto en proyectos pequeños como en grades proyectos de desarrollo, como:

- Sitios con PHP como base a nivel internacional: Wikipedia, Flickr, Yahoo (answers), Sourceforge, Digg, Delicious, y destaca sobre el resto Facebook que ha desarrollado una máquina virtual PHP (HHVM: *Hip Hop Virtual Machine for PHP*),
- Sitios PHP de importancia nacional: atrapalo, meneame.

Además de los sitios anteriores, multitud de FW y herramientas muy populares en el desarrollo web se han desarrollado en PHP:

- Los principales CMS (*Content Manager System* o Gestor de Contenidos): Joomla, Drupal, BitWeaver, DragonFly, WordPress
- Tiendas vituales: Prestashop, Magento
- Campus virtuales: Moodle
- Frameworks: Symfony, Yii Spring, Laravel, Phalcon, CodeIgniter, CakePHP, gvHidra

La flexibilidad del lenguaje ha sido durante muchos tiempo utilizado en su contra, PHP permite insertarse dentro del código HTML, de manera que depurar y/o factorizar una página web desarrollada en PHP sin el apoyo de algún FW o con un mínimo de cuidado en separar la presentación de la lógica de negocio puede convertirse en una tarea muy complicada (sobre todo el mantenimiento). Pese a que el código mal diseñado no es inherente a este lenguaje, es cierto que sus características, pueden hacer que sea propenso no es estructurado.

Para evitar estos problemas, lo mejor es hacer uso de patrones de diseño y/o algún *framework* de desarrollo que mantenga la separación entre las distintas capas de desarrollo.

[1] El nombre fue evolucionando a lo largo del tiempo, Personal Home Page Tools, Forms Interpreter, Personal Home Page Construction Kit, es con la publicación de PHP3 en Junio de 1998 cuando se designa este nombre y arranca el crecimiento explosivo del lenguaje, junto con Rasmus, colaboran los israelíes Andi Gutmans y Zeev Suraski (Zend Technologies).

1.2.2 Bases del lenguajes

El código PHP se escribe en un documento de texto al que se debe indicar la extensión *.php*. En realidad se trata de una página web HTML normal a la que se le añaden etiquetas especiales que son interpretadas por interprete o motor. Esas zonas del documento se resaltan a través de la etiqueta:

```
<?php
    echo “Bienvenidos al curso <strong>GvHidra</strong>”;
?>
```

1.2.2.1 Comentarios

En bloque (varias líneas)

```
<?php
   /*
     Comentario en
     varias líneas
   */
   $x=10;
?>
```

Comentario en línea

```
<?php
   $x=10;//Comentario en linea
   $y=20;# Otro comentario en linea
?>
```

1.2.2.2 Sintaxis

- PHP se basa en C y Perl, tiene en común con estos lenguajes:
- Todas las líneas de código deben de finalizar con un punto y coma
- Se puede agrupar el código en bloques que se escriben entre llaves
- Una línea de código se puede partir o sangrar (añadir espacios al inicio) a voluntad con el fin de que sea más legible, siempre y cuando no partamos una palabra o un valor.
- PHP es CASE Sensitive, obliga a ser estricto con las mayúsculas y las minúsculas (esto se aplica a nombres de ficheros, el mismo código en UNIX/Linux y Windows puede no ejecutarse igual), no es así con las palabras reservadas del lenguaje.

1.2.2.3 Salida

Aunque hay muchas funciones de escritura (para escribir en lo que será la página final) las fundamentales son `echo` y `print`.

Comentario en línea

```
<?php
   $variable = “ mundial”;
   echo “Hello Wordl”;
   print(“Hola Mundo $variable”.’ Concatenamos con .’);
?>
```

1.2.2.4 Variables

El identificador de las variables (nombre) tiene que cumplir:

- Debe empezar con el símbolo $. Ese símbolo es el que permite distinguir a una variable de otro elemento del lenguaje PHP.
- El segundo carácter puede ser el guion bajo (_) o bien una letra (no un número)
- A partir del tercer carácter se pueden incluir números, además de letras y el guión bajo
- No hay límite de tamaño en el nombre

La declaración de una variable es implícita, cuando comienza a utilizarse se crea (con los problemas que eso puede llevar)

Existen algunas variable predefinidas [php.net/manual/es/reserved.variables.php], de las cuales merecen especial atención:

- **`$_SESSION`**: Funciona a modo de *cluster*, donde podemos hacer persistir variables entre hilos de ejecución de PHP
- **`$_REQUEST`**: Variable superglobal estructurada en forma de *array* asociativo y que contiene la información enviada por GET, POST y COOKIE.

1.2.2.5 Operadores

Operadores aritméticos	
+	Suma dos valores - * Multiplica dos valores / Divide dos valores % Resto de dividir dos valores ++ Incremento en una unidad -- Decremento en una unidad
-	Resta dos valores (o pasa a negativo un valor)
*	Multiplica dos valores
/	Divide dos valores
%	Resto de dividir dos valores
++	Incremento en una unidad
--	Decremento en una unidad

Operadores lógicos	
!	Negación
&& and	Y
|| or	O
xor	O exclusiva

Operadores condicionales	
==	Igualdad

===	Igualdad estricta (tipo)
!=	Distinción
!=	Distinción de valor y tipo
> >=	Mayor y mayor o igual
< <=	Menor o menor o igual

Operadores de asignación	
=	Asignación
+= -=	Suma y asigna o resta y asigna
*= /=	Multiplica y asigna o divide y asigna
.=	Concatena y asigna

Además de los anteriores, tenemos el operador “referencia” & que permite que desde dos variables se apunte al mismo contenido o valor.

1.2.2.6 Tipos de datos

- Númericos: Enteros y reales o flotantes
- Cadenas
- Booleanos
- Estructuras, vectores/arrays y objetos

Estructuras de control y bucles

- IF

```
if(condición/es)
{
   acción a realizar;
}
else
{
   acción a realizar en caso de que no se cumpla;
}
```

- SWITCH

```
switch(expresión)
{
   case valor1:
     sentencia a ejecutar cuando la expresión tiene como valor valor1
   break;
   case valor2:
      sentencia a ejecutar cuando la expresión tiene como valor valor2
   break;
   case valor3:
     sentencia a ejecutar cuando la expresión tiene como valor valor3
   break
   default:
     sentencia que se ejecutar cuando no se cumpla ninguna de las condiciones anteriores
}
```

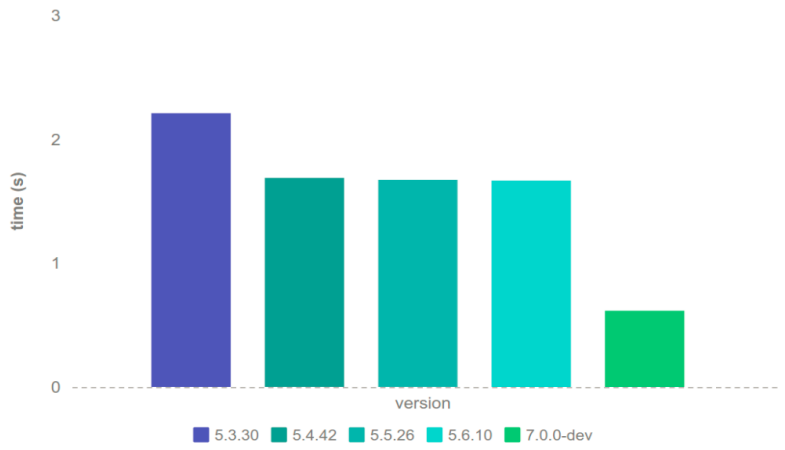
- FOR

```
for(inicialización;condición;actualización)
{
   sentencia a ejecutar mientras se cumpla la condición
}
```

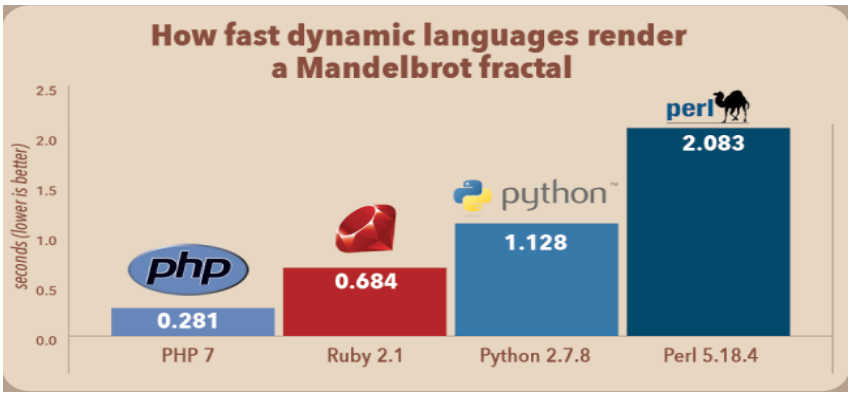
1.2.3 La evolución: PHP7

Abrimos un punto mencionando la versión PHP7 que está disponible desde finales del 2015, y que ha traído muchísimas novedades y mejoras con respecto a la versión PHP5.6, de hecho, se pasó por alto el nombre PHP6 y se salto hacia PHP7 precisamente para resaltar la cantidad de cambios.

- El nuevo motor de PHP 7, llamado PHPNG (*Next Generation*) mejora el rendimiento y lo hace comparable al PHP HHVM de facebook. Las razones principales son el uso de memorias del *Zend Engine*, así como a la introducción de un compilador JIT (*Just in time*) antes de ejecutar el código. La comunidad indica que la versión PHP7 es según indican los *benchmarks* casi el doble de rápido que las versiones 5.4 y 5.6:



- Pero el rendimiento no mejora sólo con respecto a versiones anteriores del lenguaje, sino, que si lo comparamos con otros lenguajes interpretados, vemos, las pruebas indican que el rendimiento de PHP7 es muy superior al de otros lenguajes:



- Se mejorá en el tipado de variables
- Nuevos operadores (operador de fusión de null, operador nave espacial…), constantes con arrays, clases anónimas, escapes de puntos unicode, clausuras, previsiones o aserciones, mejoras en el uso de espacios de nombres, …

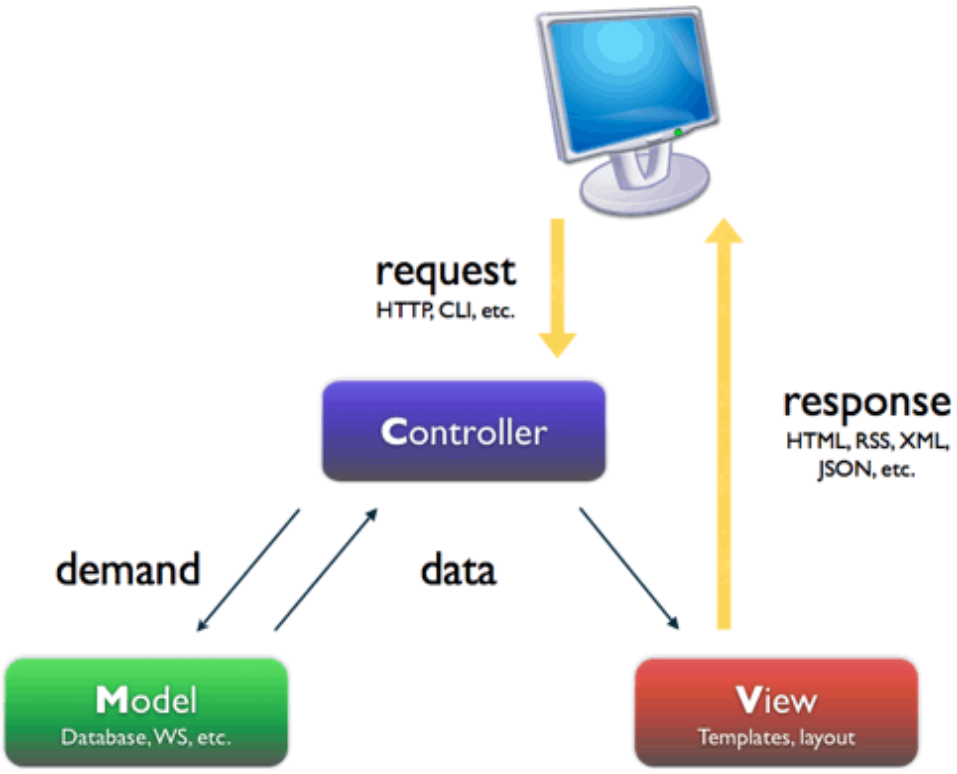
Hay más información en:

- http://php.net/manual/es/migration70.new-features.php
- http://php.net/manual/es/migration71.new-features.php
- http://php.net/manual/es/migration72.new-features.php
- http://www.zend.com/en/resources/php7_infographic

1.3 Arquitectura MVC

Es un patrón de diseño que trata de estructurar las aplicaciones en tres capas que recogen comportamientos distintos. Tiene su origen conceptual en los años 80, y vuelve a cobrar protagonismo a raíz de la evolución de las aplicaciones web.

- El **modelo** gestiona la información del sistema, permite o no acceso a la misma (consultas y actualizaciones. Sirve a la vista la información para que esta la presente al usuario.
- El **controlador** gestiona la respuesta ante eventos del usuario (selecciones en un menú, pulsación de botones, perdidas de foco. Traduce esas acciones de usuario en peticiones al modelo
- La **vista** se encarga de presentar la información del modelo y de recoger la información introducida por el usuario.



Actualmente los principales *frameworks* de desarrollo de aplicaciones (tanto en PHP, como en otros lenguajes) incorporan este patrón como base de su arquitectura.

2 Infraestructura

En este apartado vamos a distinguir la infraestructura necesaria para **ejecutar** aplicaciones gvHidra (en realidad, aplicaciones PHP) y la infraestructura necesaria para **desarrollarlas**.
Por necesidades de ejecución, entendemos el área propia de sistemas, que comprende servicios como un servidor Web (con PHP) y un SGBD principalmente.
En cuanto a la infraestructura de desarrollo, entendemos que es la propia de dicha área, un IDE (Eclipse), un cliente SVN, un cliente de BD que nos permita interrogar a los distintos SGBDs (SQuirrelSQL, PgAdmin3, SQLDeveloper...), el diseñador de informes iReport, etc.

En cualquier caso, vamos a tratar de elegir SW modificado para ser "portable" (portableApps.com), de forma que podamos llevarnos nuestro entorno completo en un *pendrive* al acabar el curso. Razón por la cual, antes de centrarnos en la instalación de las infraestructuras de servidor y/o desarrollo, vamos a instalar la plataforma PortableApps que contiene soluciones en para ambas orillas del rio.

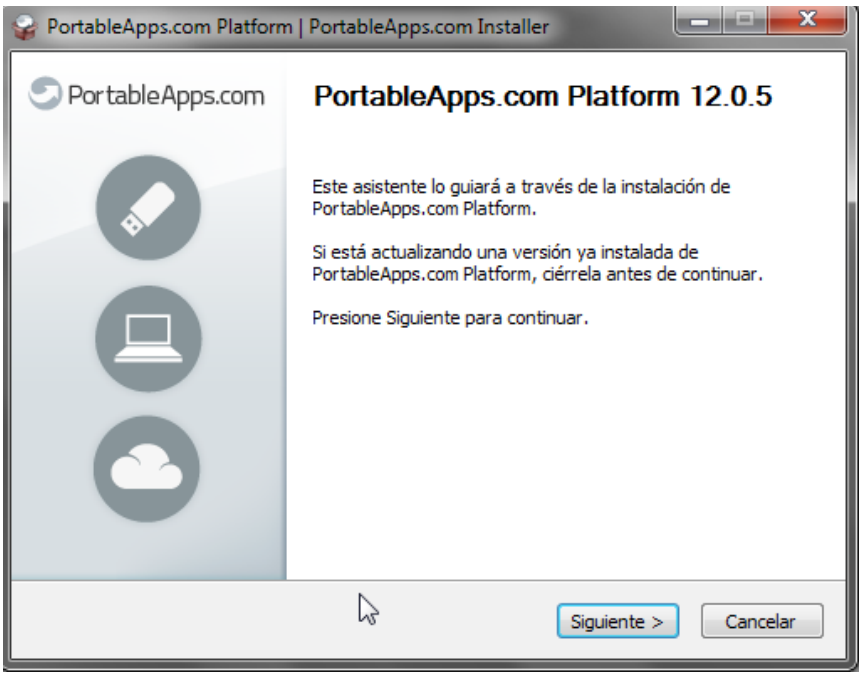
2.1 Instalación de portableApps

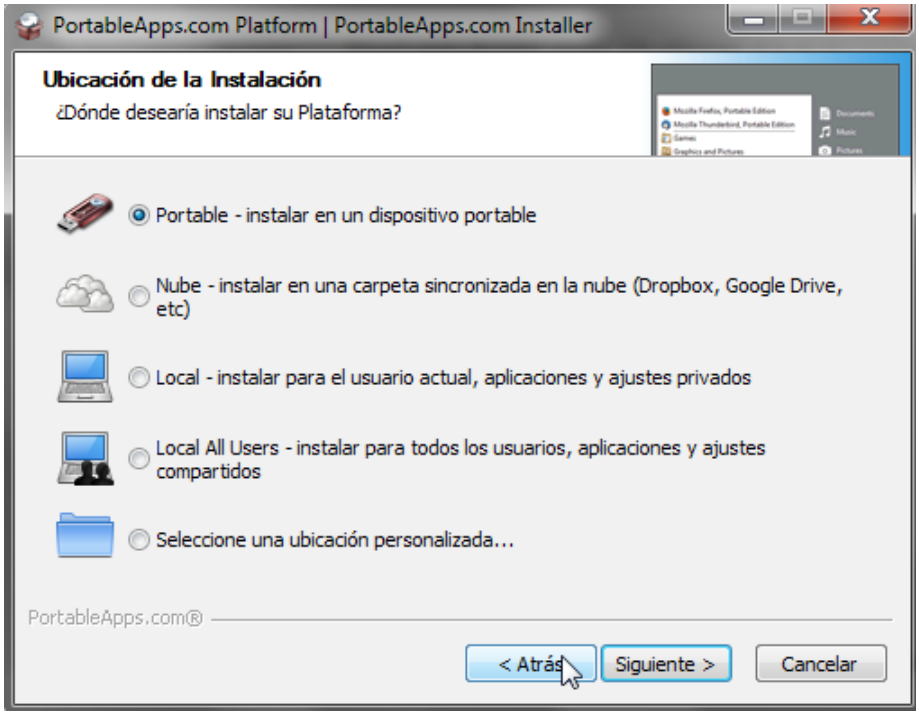
El proyecto portableApps, tiene como objetivo facilitar la instalación de aplicaciones Windows en unidades portables, o incluso en la nube (DropBox, etc.).
La elección (arriesgada) de esta plataforma para la formación de este curso es para intentar que los alumnos que lo deseen puedan llevarse a casa un entorno de desarrollo gvHidra COMPLETO incluyendo todos los ejemplos que se desarrollen durante el curso.

Lo primero es acudir al sitio web: http://portableapps.com/ y descargar la plataforma de instalación (que ubica todos sus subproyectos en sourceforge). La URL de descarga es:

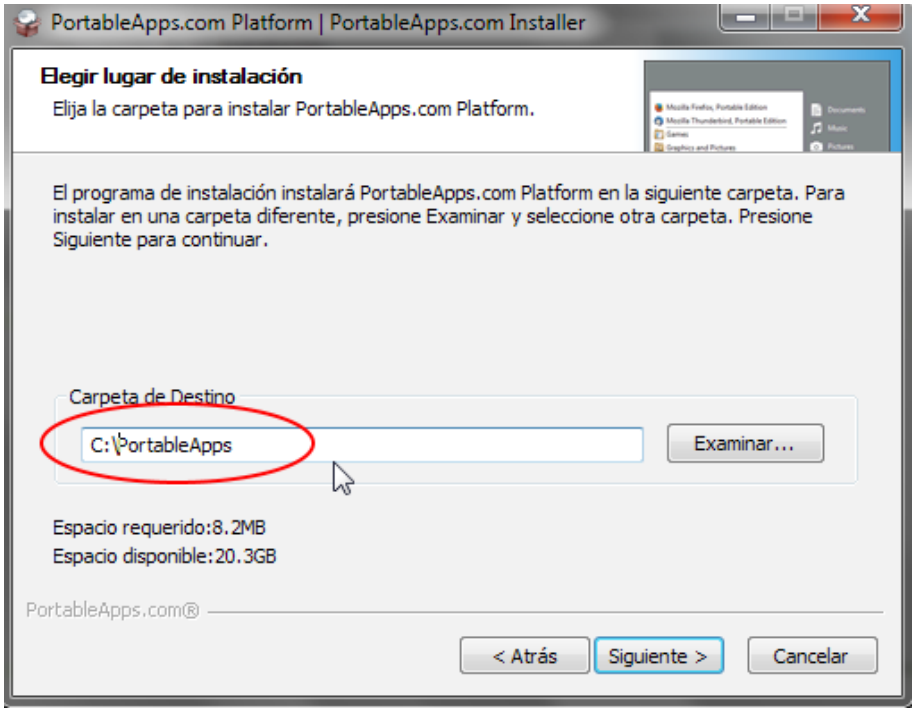
http://portableapps.com/download

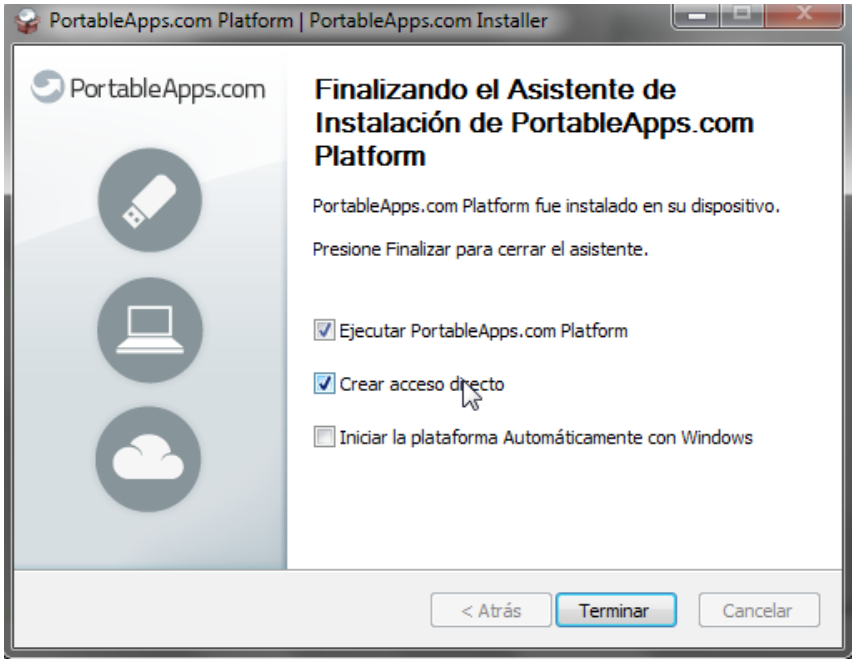
Tras la descarga, procedemos a instalar la plataforma de lanzamiento de aplicaciones portables (desde la misma podremos instalar, iniciar y actualizar las aplicaciones portables que necesitemos).



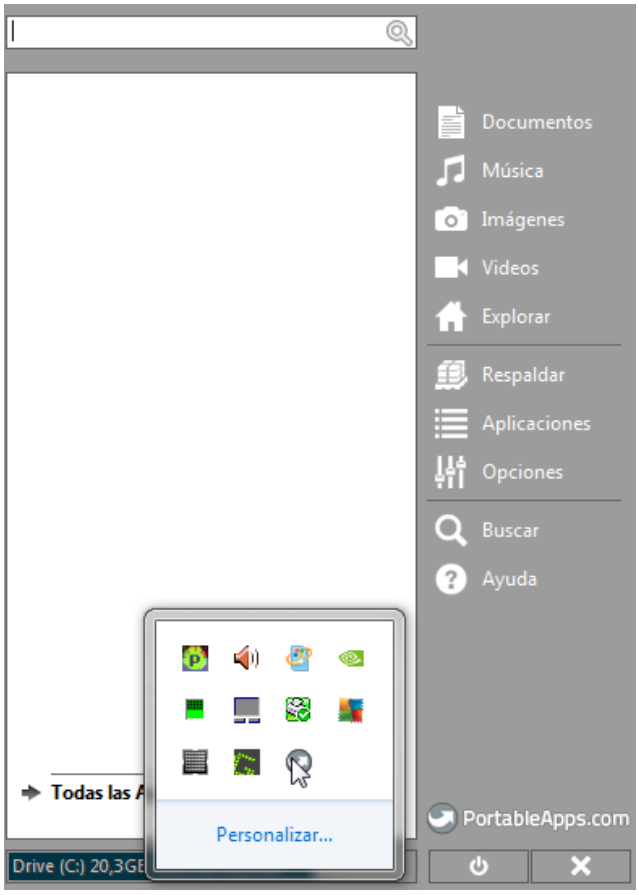


Elegimos como directorio de instalación **C:\gvh\PortableApps**





Tras la instalación, podemos arrancar la plataforma desde la barra ubicada al lado del reloj:



Si tras el arranque nos aparece un mensaje con SW para instalar vía *online*, no seleccionaremos ningún paquete y pulsaremos siguiente.
Continuaremos con la instalación de portableApps más adelante.

2.2 Entorno de servidor

En este apartado trataremos aspectos en los que no siempre se abordan en el desarrollo, por ejemplo, en la DGTIC se encarga de ello el servicio de sistemas. Vamos a explicar como convertir nuestro PC en un servidor de aplicaciones PHP. Normalmente, en la instalación de servidores, se opta por sistemas operativos GNU/Linux, pero el aula de este curso sólo dispone de entorno Windows, así que el manual se centra en una instalación para este sistema operativo. Los ficheros de configuración será similares a los de un servidor GNU/Linux (salvando rutas y la extensión de algún fichero).

Ya hemos visto en el punto anterior el uso de la plataforma portableApps, la intención es que la mayor parte del SW utilizado durante el desarrollo del curso se lo pueda llevar el alumno en una unidad de disco externa o pendrive y que el mismo siga ejecutándose.

Para ejecutar una aplicación gvHidra, necesitamos un servidor Web con soporte PHP5 y configurado para conectarse a distintos SGBDs (Oracle, PostgreSQL, MariaDB...). Para tener un entorno total, instalaremos también un SGBD relacional al que conectarnos de forma local.

Para nuestra instalación en entorno Windows, podemos realizar una instalación de un servidor Web (Apache por ejemplo), instalar PHP5 y posteriormente configurar ambos SW para que interactúen, sin embargo, resulta más práctico utilizar alguno de los muchos proyectos que resuelven de forma unificada las necesidades de un desarrollo Web PHP al uso, siendo muy cómodos de configurar y utilizar.
Entre muchas opciones, podemos destacar WAMPServer, Bitnami, EasyPHP, XAMPP. En nuestro caso, vamos a usar **UniformServer**, concretamente su serie *Zero*, elegimos este *stack* debido a:

- Es portable[2] (se puede instalar en un *pendrive* o disco externo)
- Es fácilmente configurable (el resto también lo son).
- Es muy sencillo cambiar de versión de PHP para hacerlo coincidir con la versión de algún entorno concreto. Por ejemplo DESA, PRE y/o PRO pueden no coincidir entre si, o por ejemplo, podemos tener aplicaciones fuera de los entornos NICA (Conselleria de educación, Conselleria de infraestructuras...). En otros *stacks*, se suelen centrar siempre en la última versión estable de PHP y complican un poco el *downgrade*.
    - Cuenta con paquetes de PHP5.4, PHP5.6 y PHP7
- La configuración por defecto de este *stack*, a diferencia de otros, se ciñe a las recomendaciones de PHP y nos propone una configuración bastante segura y donde se restringen opciones del lenguaje que han quedado obsoletas (*deprecated*). Ayudando a desarrollar siguiendo la evolución del lenguaje PHP y facilitando la integración.
- Cuenta con una comunidad activa, por lo que hay respuesta en caso de problemas o configuraciones particulares: http://forum.uniformserver.com/

Abordemos pues la instalación.

[2] La plataforma portableApps.com cuenta con *stacks* PHP para incluir como aplicaciones de la misma también portables.

2.2.1 Instant Client Oracle

Para poder conectar el PHP al SGBD Oracle, es necesario realizar la instalación del cliente Oracle en el entorno Windows. La instalación del cliente NO es portable, pues es necesario que se incluya el mismo en el PATH de Windows.

Para instalar el cliente Oracle debemos estar registrados en su webyacceder a la URL:

http://www.oracle.com/technetwork/topics/winsoft-085727.html

Descargaremos el cliente de la versión 11.2, concretamente:

- instantclient-basic-nt-11.2.0.4.0.zip
- instantclient-sqlplus-nt-11.2.0.4.0.zip

Tras la descarga, descomprimimos los paquetes en la ruta que queramos, por ejemplo en el directorio "`C:/gvh/portableApps/uniserverZ/instantclient_11_2`".

Debemos añadir al PATH de Windows dicha ruta para que PHP pueda cargar los módulos OCI:

- Con el botón derecho del ratón pulsamos sobre "Equipo/Mi PC" y seleccionamos "Propiedades"
- En el menú de la izquierda pulsamos sobre "Configuración avanzada del sistema"
- En "Opciones avanzadas", pinchamos sobre "Variables de entorno"
- En el apartado de "Variables del sistema", seleccionamos "Path" y editamos "Valor de la variable". Añadimos al final, con cuidado de no eliminar nada y separado por un punto y coma ';', la ruta de nuestra carpeta.

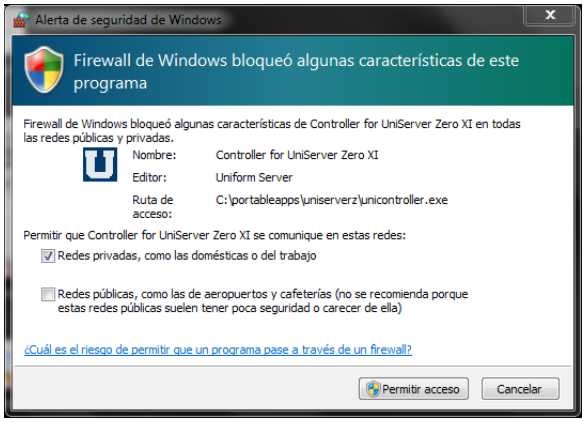
2.2.2 Servidor Web y PHP (Uniform Server)

Tras descargar el SW:

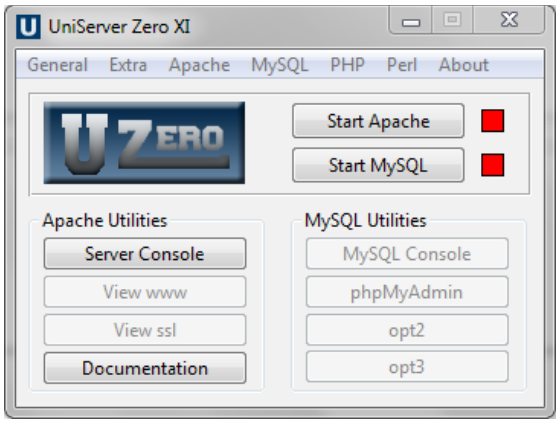
http://sourceforge.net/projects/miniserver/files/Uniform Server ZeroXI/

Simplemente debemos descomprimir el fichero "`11_5_2_ZeroXI.exe`"[3] en una ruta no alejada de la raíz de una unidad, y sin espacios en blanco, por ejemplo c:/portableApps.
Tras descomprimir ejecutaremos el fichero "`C:\gvh\PortableApps\UniServerZ\UniController.exe`".
La primera vez que lo hagamos Windows mostrará un mensaje de advertencia similar a este:

[3] La versión indicada puede ser distinta de la actual, que será la utilizada en curso, sin embargo, el proceso de instalación será similar.



Tras dar permiso tendremos acceso a un pequeño panel de control desde el que se puede llevar a cabo cualquier configuración necesaria.



Además del paquete básico, la instalación de SW adicional, así como la actualización del existente se lleva a cabo en forma de módulos, los distintos módulos se pueden descargar aquí:

http://sourceforge.net/projects/miniserver/files/Uniform Server ZeroXI/ZeroXImodules/

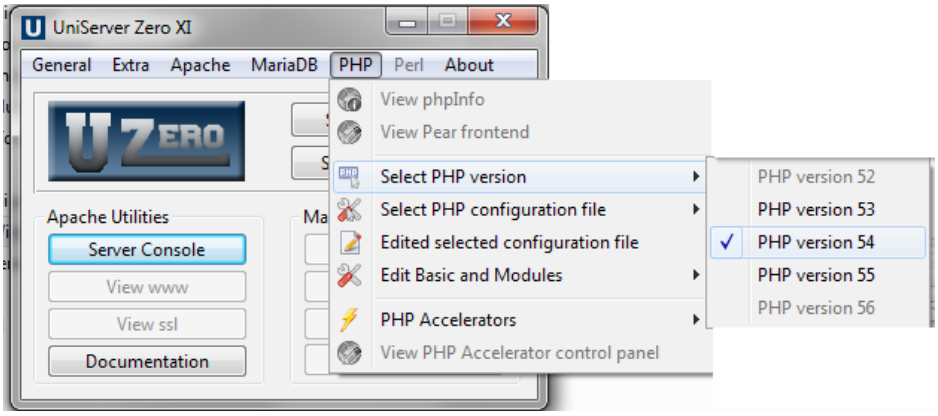
Para completar nuestro entorno, vamos a instalar los módulos siguientes:

- ZeroXI Controller
- Zero XI Documentation
- ZeroXI Adminer
- Zero XI Pear

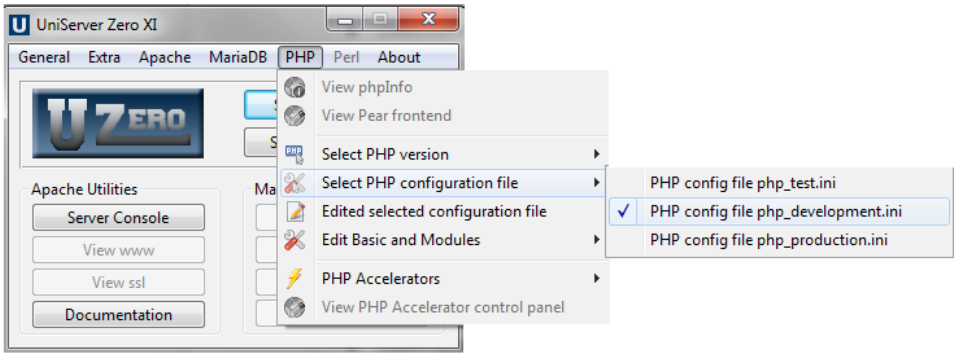
Para la instalación, basta con descargar los ejecutables de los módulos (son ficheros autoextraíbles) en el directorio del Uniform Server ("`C:\PortableApps\UniServerZ\`" en nuestro ejemplo) y descomprimimos (ejecutamos el .exe) sobrescribiendo los ficheros que nos pregunte, asegurándose de que los servicios del *stack* Uniform Server están todos detenidos.

Tras descomprimir los módulos, podremos seguir con aspectos de configuración.

- Seleccionamos la versión de PHP 5.4 (es la que viene por defecto)



- Seleccionamos el fichero de configuración de desarrollo



- Editamos el fichero de configuración PHP seleccionado y fijamos:
  - `output_buffering = 32768`
  - `max_execution_time = 120`
  - `max_input_time = 120`
  - `max_input_nesting_level = 16384`
  - `max_input_vars = 10000`
  - `memory_limit = 256M`
  - `error_reporting = E_ALL & ~E_NOTICE & ~E_STRICT & ~E_DEPRECATED`
  - `post_max_size = 32M`
  - `upload_max_filesize = 32M`
  - `max_file_uploads = 40`
  - ***`;extension=php_oci8.dll`*** `; Use with Oracle 10gR2 Instant Client`
  - ***`extension=php_oci8_11g.dll`*** `; Use with Oracle 11gR2 Instant Client`
    (De las anteriores elegir una en función del cliente instalado)
  - `extension=php_pdo_mysql.dll`
  - `extension=php_pdo_oci.dll`
  - `extension=php_pdo_odbc.dll`
  - `extension=php_pdo_pgsql.dll`
  - `extension=php_sockets.dll`
  - `extension=php_sqlite3.dll`
  - `extension=php_soap.dll`
  - `date.timezone = "Europe/Madrid"`
  - `session.name = PHP54SESSID`
  - `soap.wsdl_cache_enabled=0`

Tras ello, editamos el fichero de configuración del servidor Apache (httpd.conf) y añadimos al final:

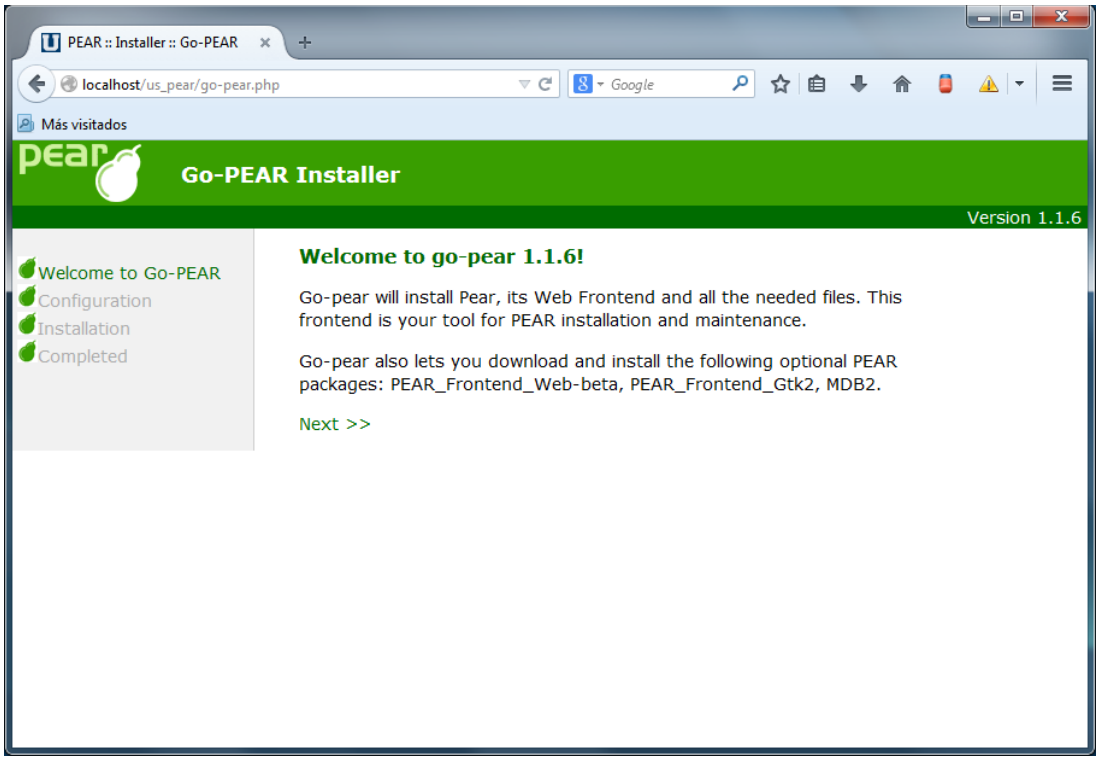
```
##########################################
# -------- VARIABLES DE ENTORNO -------- #
##########################################
#Variable que indica ruta a JRE para ejecución de JASPER
SetEnv JAVA_HOME C:\Program Files\Java...[RUTA A JRE]...
#Variables de entorno para Oracle (Si está instalado)
SetEnv ORACLE_HOME "C:/gvh/portableApps/UniserverZ/instantclient_11_2"
#SetEnv TNS_ADMIN "C:/gvh/portableApps/instantclient_11_2/tnsnames.ora"
##########################################
# ------ FIN VARIABLES DE ENTORNO ------ #
##########################################
```

2.2.2.1 PEAR

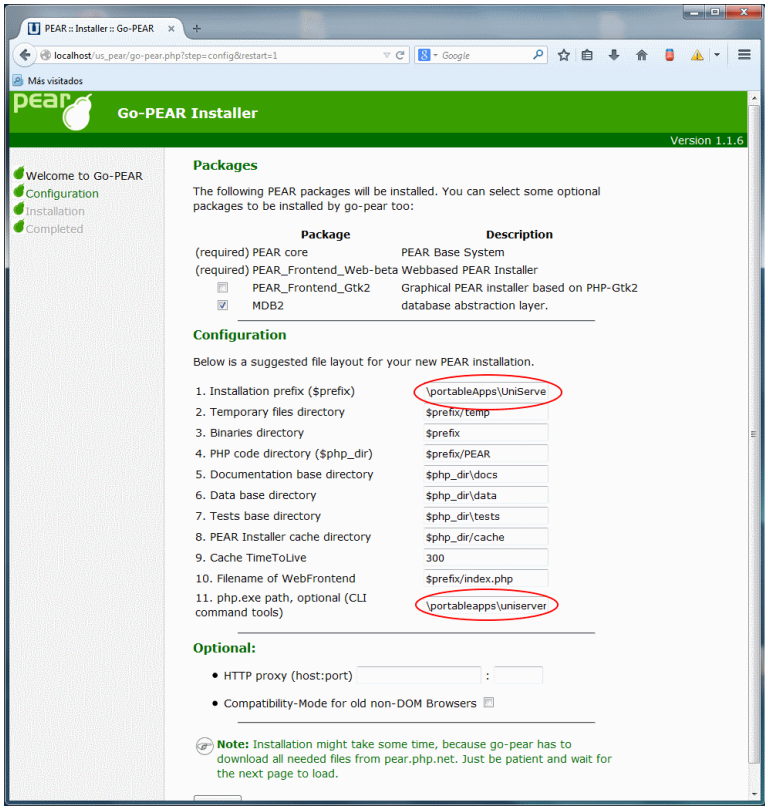
Tras la instalación y arranque, sólo queda configurar el PEAR (que habíamos instalado previamente) para que funcione de forma correcta y podamos instalar los paquetes necesarios.
Para ello, abrimos la URL:

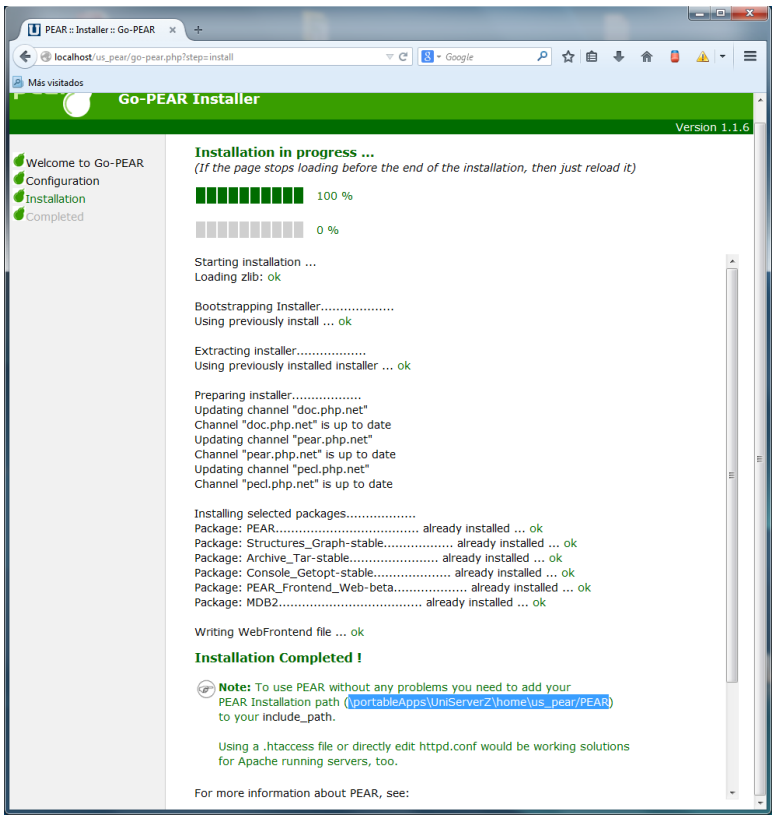
http://localhost/us_pear/go-pear.php

Y completamos la configuración según los parámetros de nuestra instalación:



Elegiremos la instalación de PEAR dejando los parámetros que aparecen por defecto, es decir, la ruta base de instalación será "**\gvh\PortableApps\UniServerZ\home\us_pear**".





Tras la instalación, debemos activar el *frontend* Web para gestionar los paquetes PEAR, para ello, editamos el fichero:

`UniServerZ\home\us_pear\index.php`

Y dejamos su contenido así:

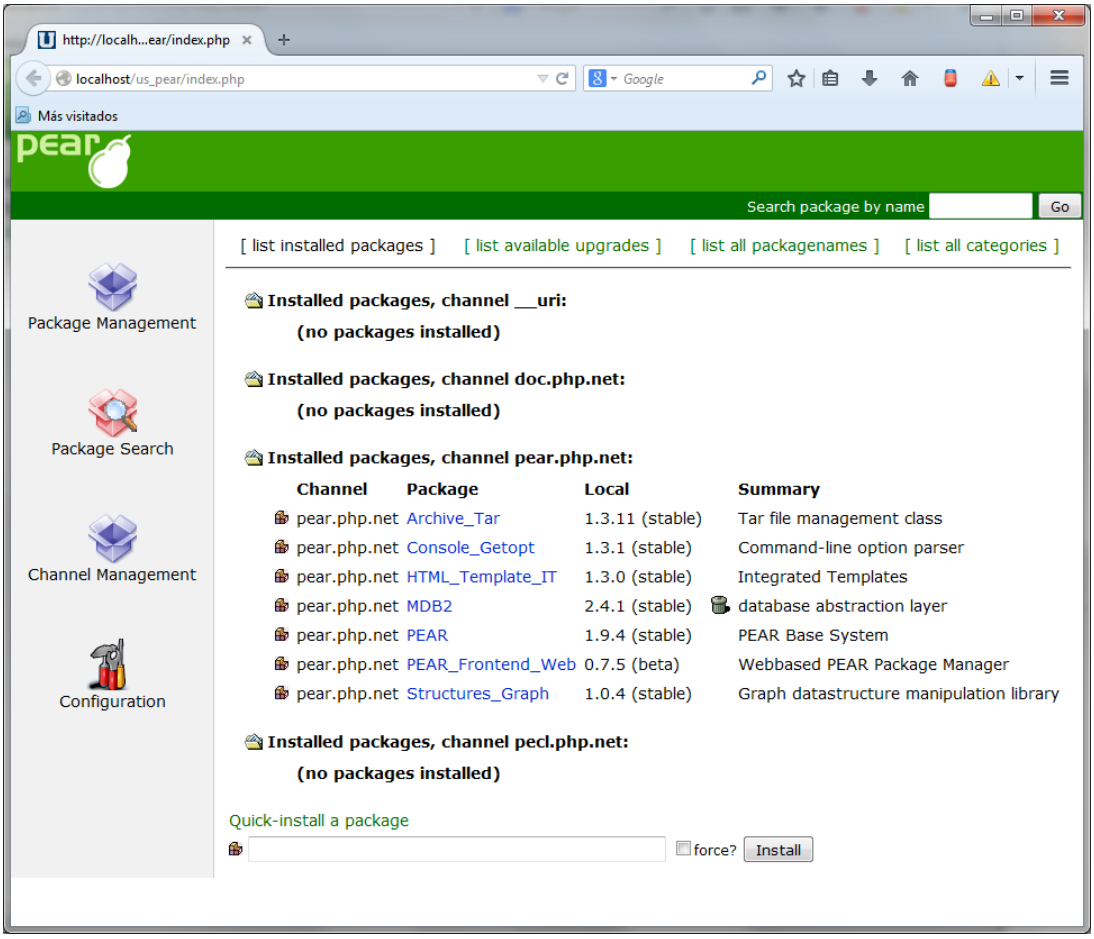
```
1 <?php
2 /**
```

```
 * Put this file in a web-accessible directory as index.php (or similar)
 * and point your webbrowser to it.
 */

$us_path = realpath('.'); // Determine full path to this file portability

// $pear_dir must point to a valid PEAR install (=contains PEAR.php)
$pear_dir = $us_path.'\PEAR'; // default of install

// OPTIONAL: If you have a config file at a non-standard location,
// uncomment and supply it here:
$pear_user_config = $us_path.'\pear.conf';

// OPTIONAL: If you have protected this webfrontend with a password in a
// custom way, then uncomment to disable the 'not protected' warning:
$pear_frontweb_protected = true;

/*********************************************************
 * Following code tests $pear_dir and loads the webfrontend:
 */
if (!file_exists($pear_dir.'/PEAR.php')) {
    trigger_error('No PEAR.php in supplied PEAR directory: '.$pear_dir,
                  E_USER_ERROR);
}
ini_set('include_path', $pear_dir);
require_once('PEAR.php');

// Include WebInstaller
putenv('PHP_PEAR_INSTALL_DIR='.$pear_dir); // needed if unexisting config
require_once('pearfrontendweb.php');
?>
```

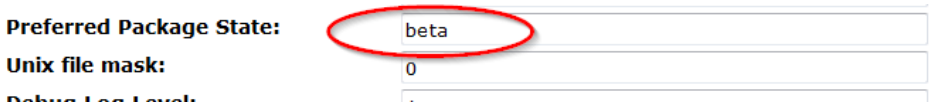
Tras ello podemos acceder a la interfaz Web de configuración de PEAR en la URL:

http://localhost/us_pear/index.php

Desde dicha URL podemos afinar opciones de configuración y/o instalar paquetes:



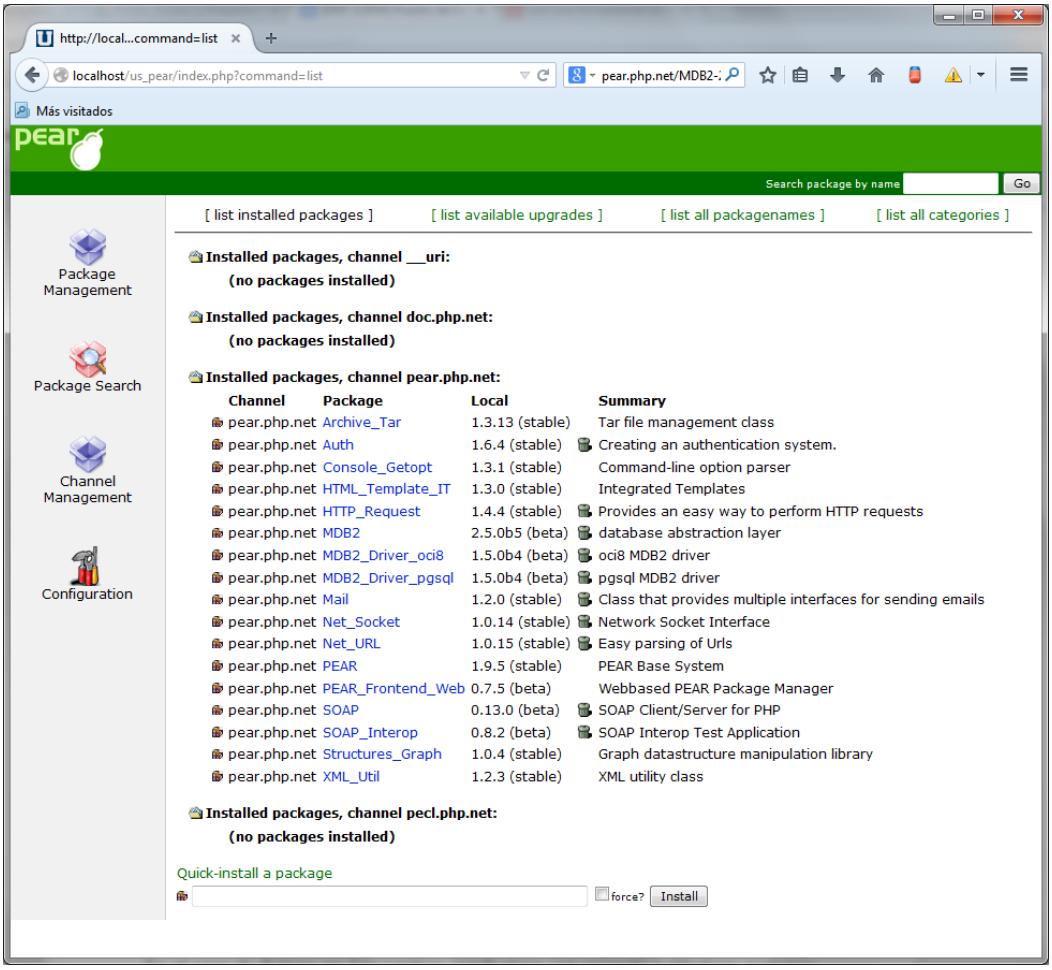
Deberemos fijar el uso de paquetes pear en beta (***Configuration***) , para que se puedan utilizar las ultimas versiones de MDB2:



Tras ello elegir la opción "***list available upgrades***" y actualizar todos los paquetes (según que versión de PEAR, puede que MDB2 aparezca como actualizable, o sea necesario eliminarlo y volver a instalar).

Además de lo anterior deberemos instalar como mínimo, los siguientes paquetes:

- `Auth`
- `Mail`
- `MDB2_Driver_oci8`
- `MDB2_Driver_pgsql`
- `MDB2_SOAP`
- `MDB2_Soap_Interop`



Si en algún momento tenemos un error de dependencias porque hay algún paquete que no se ha actualizado a la versión beta, podemos reinstalarlo y marcar la opción "***force***".

Con esto contamos con un servidor Web de desarrollo en local para trabajar cuando lo necesitemos, podemos crear nuestra primera página php e incluirla en el directorio que sirve los documentos:

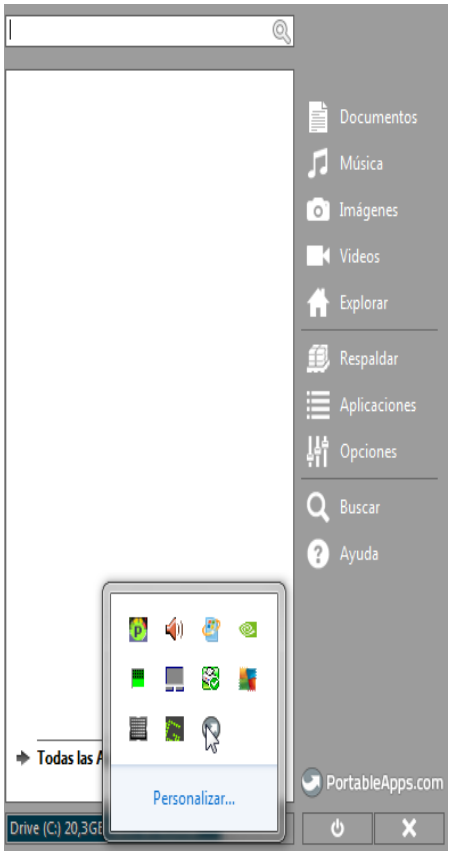
Fichero "`C:\portableApps\UniServerZ\wwwe\phpinfo.php`"

```
<?php phpinfo(); ?>
```

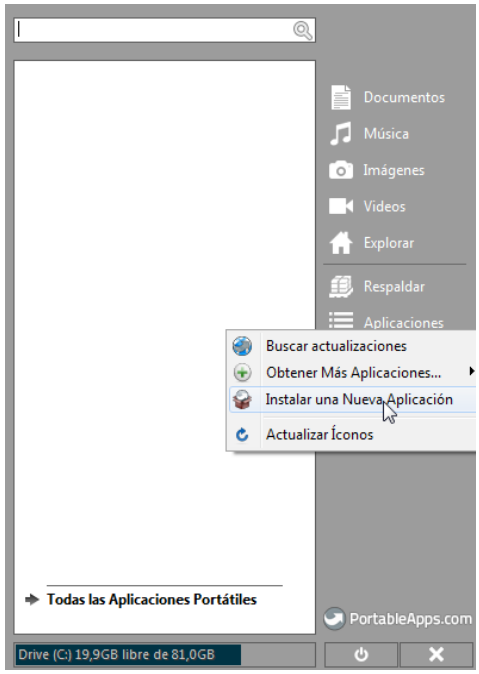
2.2.3 SGBD PostgreSQL

Para la instalación del SGBD OpenSource PostgreSQL volveremos a utilizar la plataforma portableApps, que dispone de una versión de postgreSQL que puede arrancarse de forma portable. El rendimiento de esta versión portable es muy inferior al que puede obtenerse instalando y configurando el SGBD postgreSQL como servicio del sistema, sin embargo, nos permitirá llevar a cabo algún ejercicio práctico durante el presente curso.

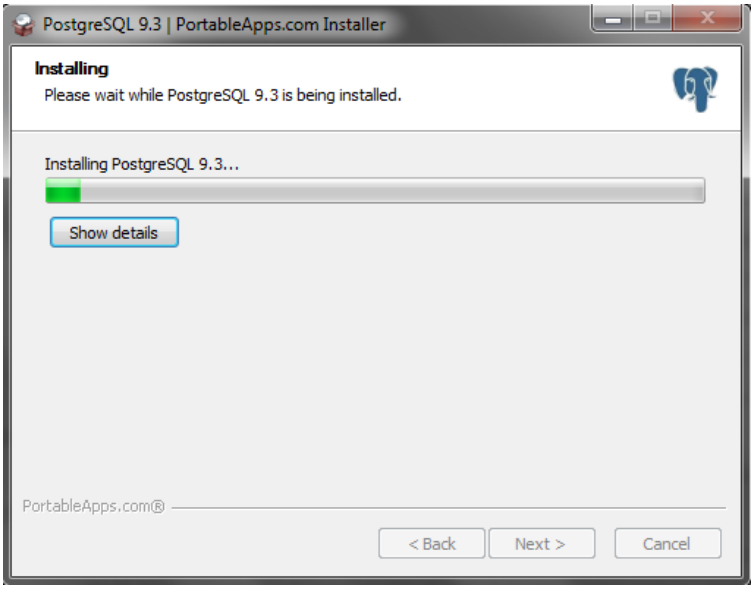
Para la instalación, arrancamos la plataforma desde el icono de al lado del reloj en la barra de tareas:



Desplegamos el menú “aplicaciones” y elegimos “instalar nueva aplicación”.



Buscamos la ubicación del SW PostgreSQL portable y procedemos a la instalación del mismo.



Tras la instalación, si decidimos ejecutar PostgreSQL, tendremos una instancia del mismo corriendo, eso si, sin configurar y con un rendimiento bajo.

Si deseamos realizar una instalación de postgreSQL al uso, podemos:

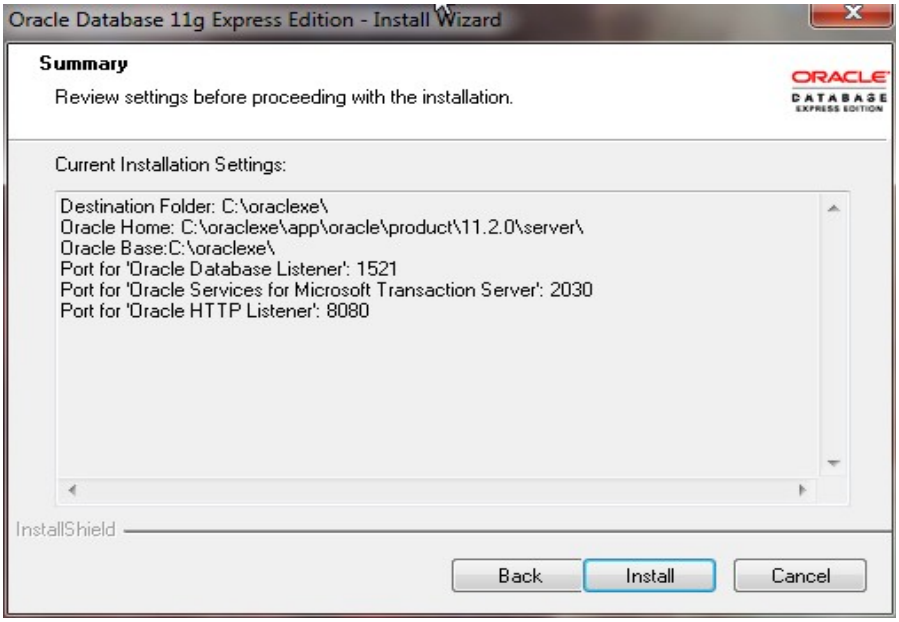
- Descargar la ultima versión estable (9.3.5) de PostgreSQL desde la URL correspondiente: http://www.postgresql.org/download/windows/
- Ejecutar el fichero de instalación dejando las opciones por defecto
- Establecer la contraseña del administrador (usuario **postgres**) como ***curso***, dejando el puerto de escucha propuesto por defecto (**5432**), la configuración regional por defecto y sin marcar la opción de instalación "Stack Builder"
- Por ultimo añadir al PATH la ruta "`C:\Archivos de Programa\PostgreSQL\9.3\bin`".

2.2.4 SGBD Oracle 11XE

Con la aparición de distintos SGBDs en el mundo *openSource* que han plantado cara a Oracle, la compañía decidió con intención de acercar a su producto a los estudiantes y desarrolladores poner a disposición de forma gratuita una versión reducida del Oracle 11, denominado Oracle 11 XE (*express edition*).
La descarga del mismo sólo requiere registrarse en Oracle, podemos descargar el SW desde aquí:

http://www.oracle.com/technetwork/database/database-technologies/express-edition/downloads/index.html

Tras la descarga, procedemos a descomprimir el SW y ejecutamos el fichero "**`DISK1\setup.exe`**". Dejaremos la ruta de instalación por defecto (`c:\oraclexe`). Como contraseña de **`sys`** y **`system`** pondremos "**curso**".



Una vez finalizada la instalación, arrancamos la base de datos con el comando ***Inicio → Oracle Database 11g Express Edition → Start database***.

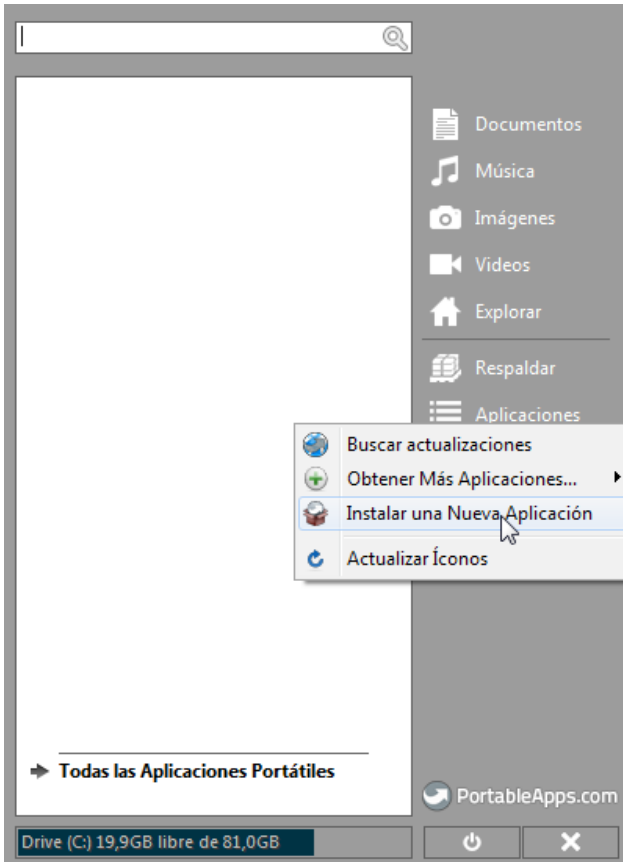
Podemos iniciar una conexión con comando “Run SQL Command Line”, tecleando:

```
SQL> connect system/curso;
```

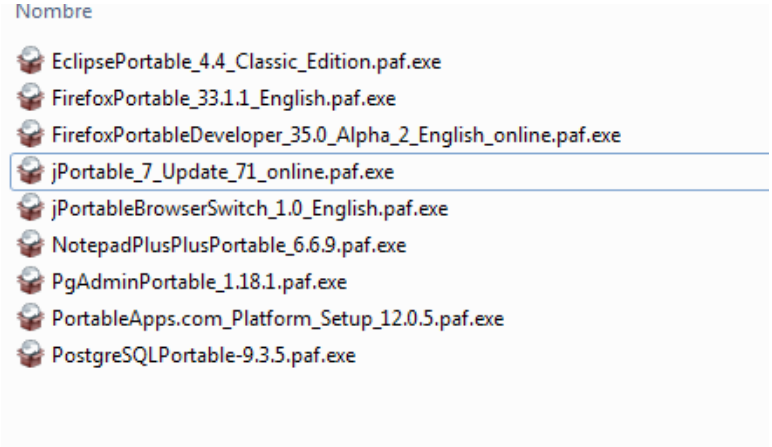
2.3 Entorno de desarrollo

2.3.1 JRE

Utilizaremos la instalación propuesta por portableApps. Para ello, y como en ocasiones anteriores, lanzamos la plataforma a través del icono del área de notificaciones de Windows (en la parte derecha de la barra de tareas, al lado del reloj) y seleccionamos la instalación de una nueva aplicación:



En este caso, el SW jPortable online:



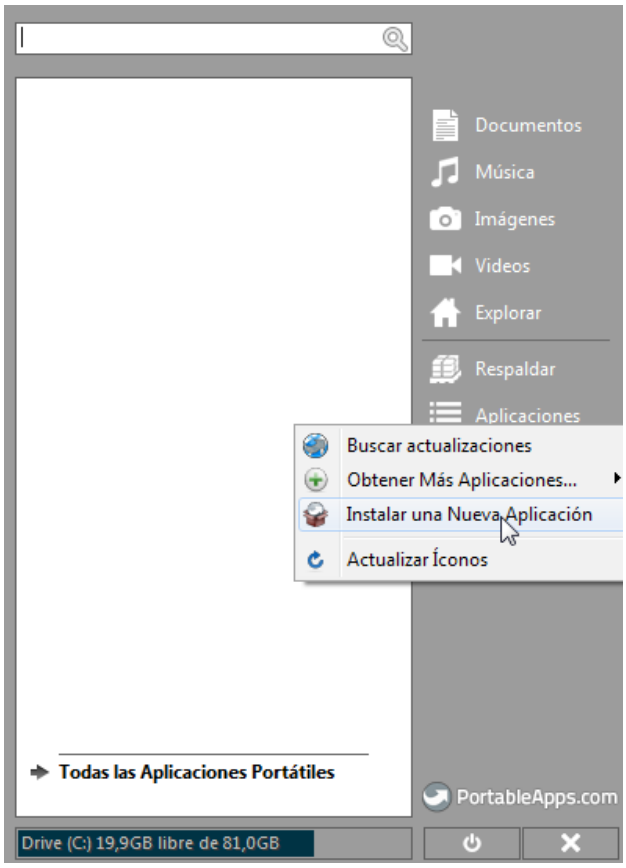
Tras ello, podemos proceder a instalar aplicaciones portables que necesiten de entorno Java para su ejecución, como es el caso de Eclipse.

2.3.2 Eclipse

2.3.2.1 Instalación base

Al igual que antes instalaremos una versión de eclipse Luna portable, para ello, y como en

ocasiones anteriores, lanzamos la plataforma portableApps a través del icono del área de notificaciones de Windows (en la parte derecha de la barra de tareas, al lado del reloj) y seleccionamos la instalación de una nueva aplicación:



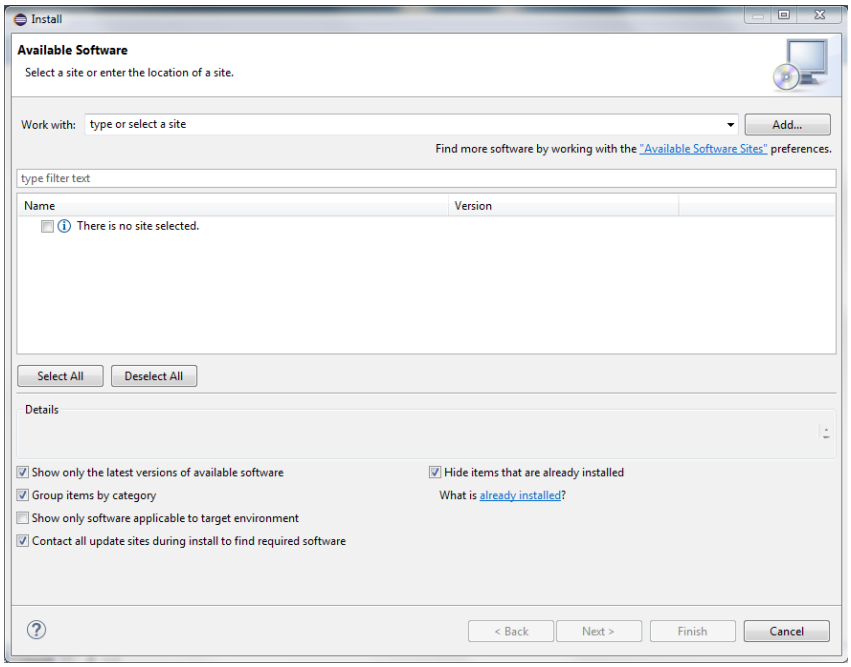
Elegimos el fichero EclipsePortable_4.4_Classic y procedemos como en ocasiones anteriores.

Si quisiéramos hacer una instalación estándar seguiríamos estos pasos:

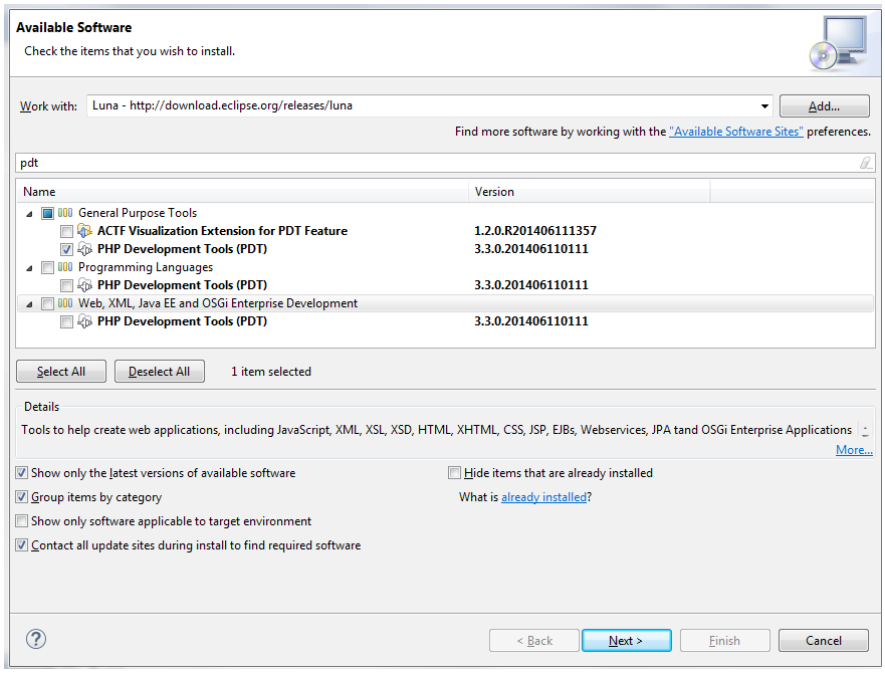
- Descarga Instalar JRE (recomendado 32 para mejorar portabilidad) de http://www.java.com/es/ e instalar.
- Descargar Eclipse (32 o 64 Bits acorde a JRE) desde https://www.eclipse.org/downloads/. Elegimos la opción base o estándar a la que posteriormente añadiremos otras opciones.
- Ejecutar el instalador

Tras el arranque de Eclipse (tanto si se trata de la versión portable, como la versión estándar) tenemos que personalizar nuestro Eclipse para desarrollos PHP/gvHidra, para ello:

- Lo primero actualizar el eclipse. “Help → Check for updates”
- Añadir a Eclipse la funcionalidad PDT (PHP Developments Tools). Para ello ir a la opción de menú “Help → Install New Software”.

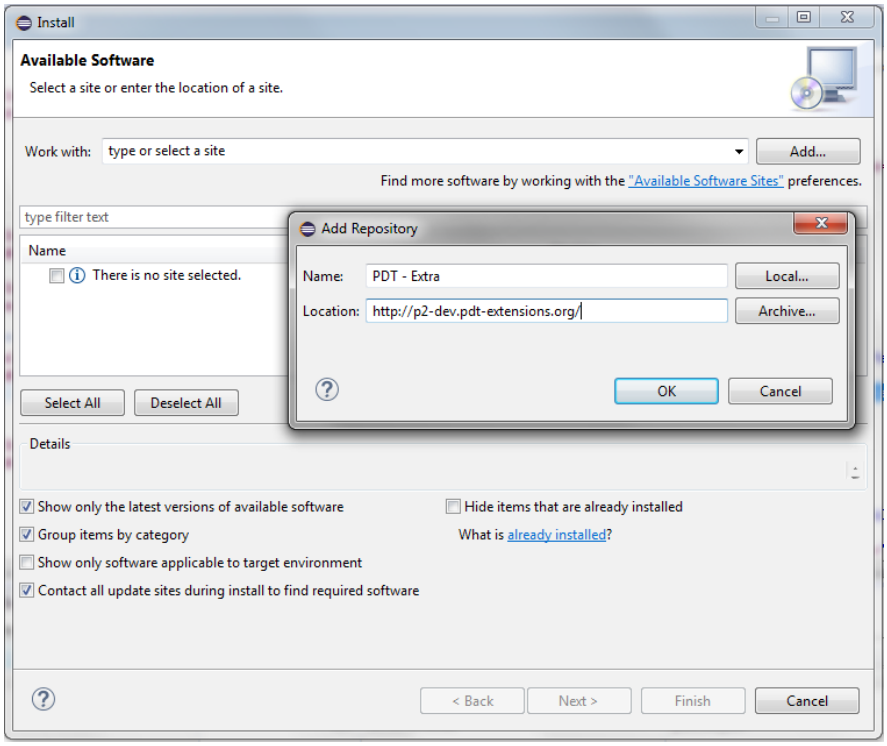


Desplegar la opción *Work with* y elegir "Luna - http://download.eclipse.org/releases/luna". Esperar a que se carguen las opciones disponibles, teclear en el filtro de búsqueda "PDT" y elegir:

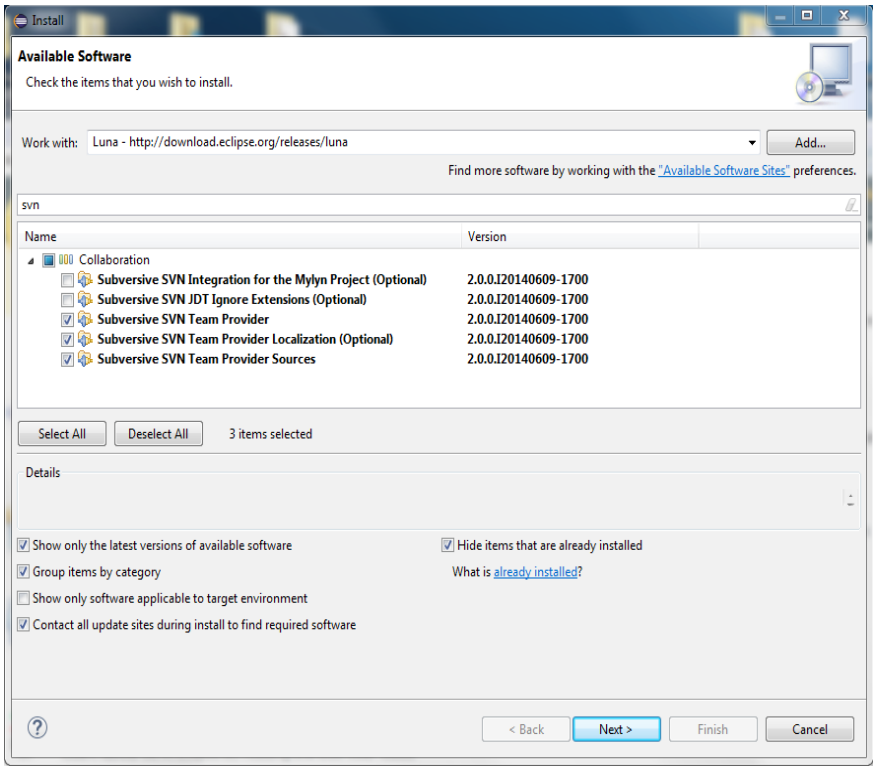


Pulsar siguiente, los paquetes serán descargados, se pedirá el reinicio de Eclipse y terminará la instalación.

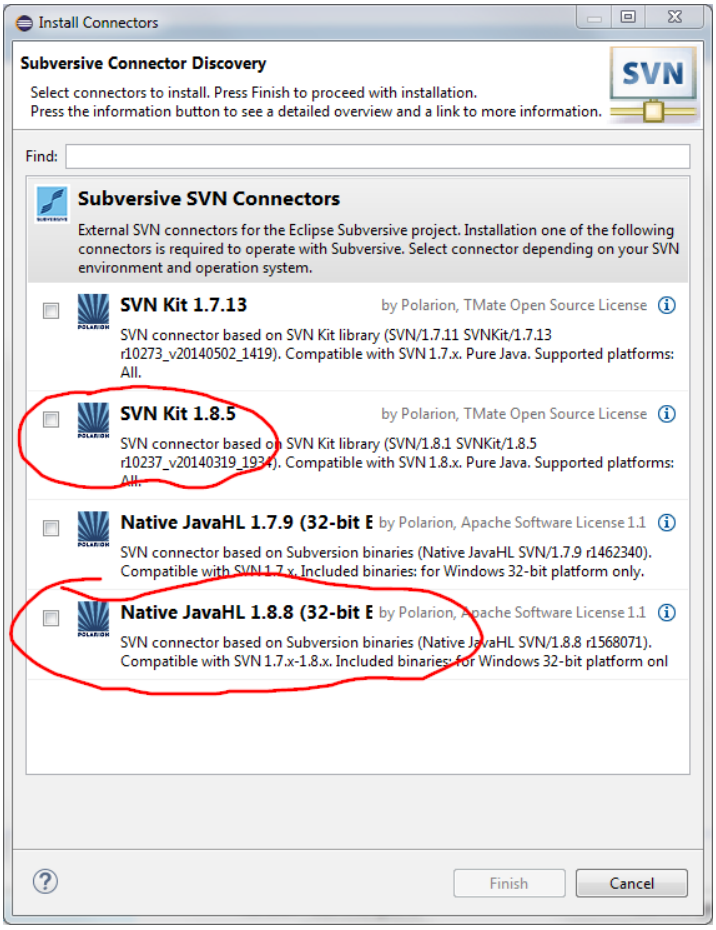
- Añadir a Eclipse la funcionalidad EXTRA de PDT (soporte a Smarty, etc...). Para ello debemos ir a la opción de menú "Help -> Install New Software" y elegir "Add" e introducir la URL: http://p2-dev.pdt-extensions.org. Tras ello, elegimos as extensiones que deseemos (Smarty)



- Añadir a Eclipse la funcionalidad de cliente SVN. De nuevo, volvemos a la opción de menú "*Help → Install New Software*". Elegimos el repositorio Luna, Buscamos la cadena "*svn*" y marcamos las opciones de la figura.



Tras la instalación cuando abramos por primera vez la vista SVN (*Window → Show View → SVN Repositories*) aparecerá un diálogo de configuración de conectores SVN similar al de la figura siguiente. Elegimos las versiones SVN Kit 1.8 y Native Java HL.



2.3.3 Firefox y plugins de desarrollo

Si un navegador ha marcado a gvHidra ha sido Mozilla Firefox. Un navegador multiplataforma que ha evolucionado siguiendo las ultimas tendencias, respetando y adaptándose a los estándares Web en constante evolución (CSS, HTML y Javascript).
Dando muy buen soporte al desarrollador (herramientas de debug, plugins, etc...).

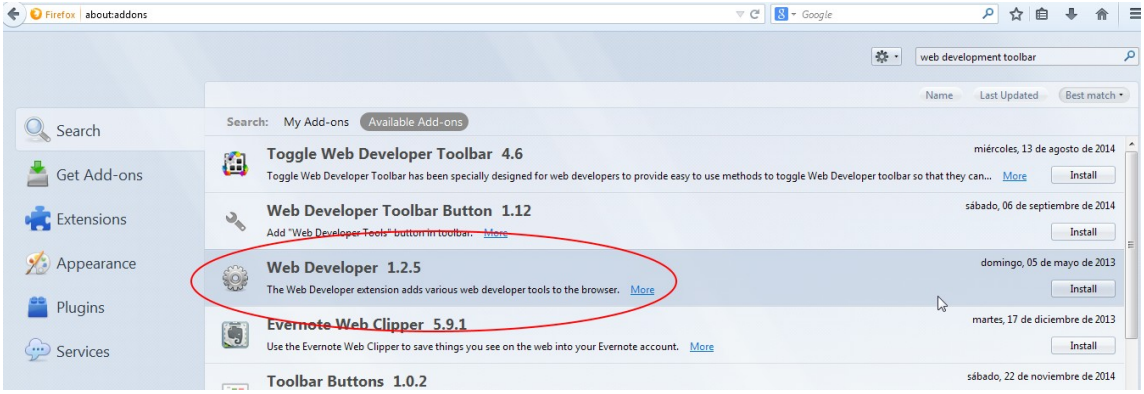
A pesar de que actualmente tiene un duro competidor en Chrome (el navegador de Google) sigue estando en primera línea de fuego.

Para su instalación podemos optar por utilizar el paquete portable (precederíamos de de forma similar a otros instalaciones de SW portable detalladas en el presente manual) o bien descargar directamente se la página web: https://www.mozilla.org/es-ES/firefox/new/

Sea cual sea nuestra opción, para desarrollar aplicaciones gvHidra pueden ser útiles un par de plugins que presentamos a continuación:

- La extensión Web Developer Extension (disponible también para Chrome)
- La extensión HTML Validator

Para instalar la primera podemos hacer uso del apartado *ad-dons* de Firefox y buscar la misma en su market.



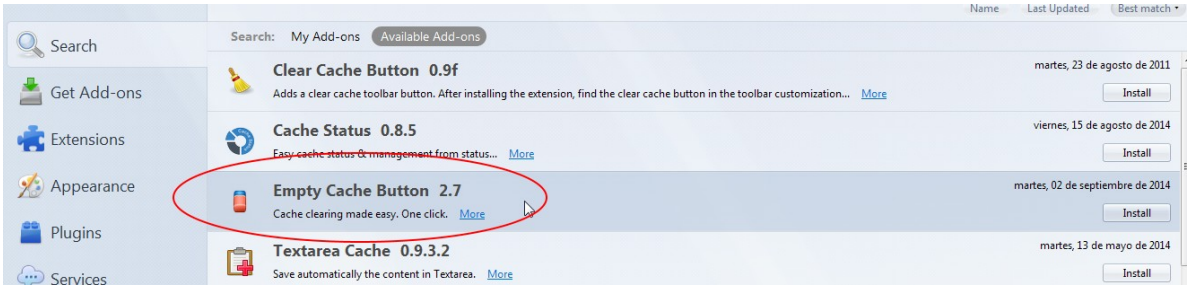
La segunda extensión es más sencilla de localizar si acudimos a la web del desarrollador y la instalamos desde allí:

DOWNLOAD

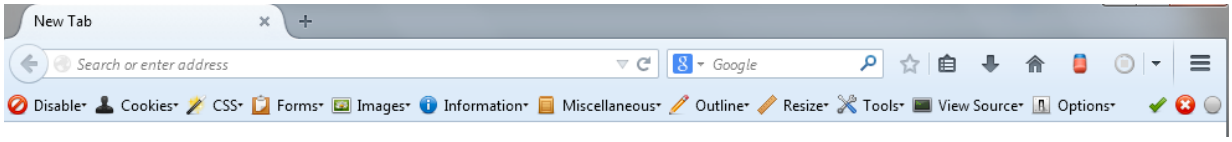
FIREFOX 23.0x or later Choose your platform :

	Version		4.0	...	23.0 and later
Windows	0.958	Download HTML Validator for Windows 32 bits	X	X	X
Linux 32 bits	0.958	Download HTML Validator for Linux 32 bits	X	X	X
Linux 64 bits	0.958	Download HTML Validator for Linux 64 bits	X	X	X
Mac Os X (universal build)	0.958	Download HTML Validator for Mac OS X	X	X	X
Other platforms		Please check the forums			
Source code	0.958	Download the source code			

Además de las anteriores, puede ser útil también algún botón o acceso directo para eliminar la caché del navegador, de estos hay multitud disponibles en el *market* de Firefox, podemos elegir cualquiera.



Tras la instalación, nuestro navegador tendrá un aspecto similar a este:



2.3.4

2.3.5 iReport

La herramienta de diseño de informes iReport, nos permite confeccionar y ejecutar de forma visual un reporte (informe) basado en el motor de reportes JasperReport.

La necesidad de contar con esta herramienta, es porque los listados y reportes en gvHidra se realizan principalmente a través de este motor de plantillas, vía el proyecto satélite jasper, que permite integrar esta tecnología java con las aplicaciones gvHidra PHP.

Recomendamos la descarga e instalación del paquete iReport 5.1.0 (actualmente la versión estable es la 5.6.0 y el proyecto ha evolucionado hacia la plataforma JasperStudio).

La descarga e instalación de esta herramienta no puede realizarse a través de portableApps, por lo que descargaremos e instalaremos la misma a mano[4]. La descarga puede realizarse desde la URL:

http://sourceforge.net/projects/ireport/files/iReport/

Se han detectado en algunos listados problemas con el tamaño de la fuente al recompilar listados desde versiones anteriores, por ello, en el momento de escribir estas líneas recomendamos la intalación de la versión 5.1.0 del mismo.

http://sourceforge.net/projects/ireport/files/iReport/iReport-5.1.0/

2.3.6 SquirrelSQL

SquirrelSQL Client es un cliente universal de bases de datos que permite conectarse a cualquier base de datos siempre y cuando ésta tenga un controlador JDBC. A través del mismo, podemos conectarnos entre otras muchas a Oracle, MySQL, PostgreSQL, DB2, HSQLDB, Sybase, Informix
Al estar programado en java, podemos ejecutarlo tanto en Windows como en plataforma GNU/Linux.

Con SquirrelSQL se pueden realizar consultas SQL tanto de definición de datos (DDL) así como también de control de datos (DCL) y tener conexiones establecidas contra distintos SGBDs desde una misma interfaz de usuario.

Si bien la herramienta no suele servir para los DBAs o administradores porque no explota todas las características del SGB al que se conecta, si suele ser de gran ayuda durante el desarrollo de aplicaciones con gvHidra, pues permite conectarnos tanto a Oracle como a PostgreSQL desde una sola aplicación (la otra opción es utilizar PGAdmin3 para postgreSQL y SQLDeveloper para Oracle)

Podemos descargar e instalar el SW desde: http://squirrel-sql.sourceforge.net/

[4] Puede que nos obligue a realizar una instalación de JRE/JDK para todo el sistema, pues está realizada en Java.

3 El *framework* gvHidra

3.1 Historia

A pesar de la ingente información publicada, sobre gvHidra a través de los años, siempre presentamos la herramienta de la misma forma:

<<gvHidra son las iniciales de Generalitat Valenciana: Herramienta Integral de Desarrollo Rápido de Aplicaciones. Nace al liberar el proyecto interno IGEP (que son las iniciales de Implementación de la Guía de Estilo en PHP) como software con licencia GPL.>>

Si tratamos de dar una definición técnica, podemos decir que se trata de, un entorno de trabajo (*framework*), para el desarrollo de aplicaciones de **gestión** en entornos web, con PHP. Que sigue una guía de estilo (una guía para unificar los criterios de aspecto y usabilidad en el proceso de desarrollo de aplicaciones). Es una herramienta RAD (*Rapid Application Development*) que se adapta a las necesidades particulares de las aplicaciones de las grandes organizaciones.

Siguiendo la definición de Roger S. Pressman en su libro Ingeniería del Software de framework, podemos decir que gvHidra *<<es un esqueleto con una colección de puntos de conexión que le permiten adaptarse a un dominio de un problema específico. Estos puntos permiten integrar clases o funcionalidades específicas del problema. En conclusión, es un conjunto de clases que cooperan>>*.

El Servicio de Organización e Informática (SOI) de la Conselleria de Infraestructuras y Transporte (en adelante CIT) emprendió en 2007 el proyecto gvPONTIS cuyo objetivo principal era migrar todos los sistemas a sistemas de código abierto. Entre otras facetas afectadas se encontraban los entornos de programación donde se utilizaba Oracle Forms y PowerBuilder, que se deciden migrar paulatinamente a lenguajes *opensource*, como eran PHP y Java.
También se apostó por la migración del SGBD Oracle hacia un SGBD abierto como PostgreSQL, y pasar a una arquitectura de aplicaciones basada en arquitectura cliente servidor a desarrollos web 3 capas para optimizar trabajos de microinformática (simplificación del puesto de trabajo y compatibilidad en MS Windows y GNU/Linux)

Con todo ello, se inicia el arranque de un proyecto en PHP que, basándose en una guía de estilo de las aplicaciones de la CIT (con la cual ya se habían hecho un par de aplicaciones piloto en PHP), aportará valor a los desarrollos aumentando la productividad de los mismos. Así nace el proyecto IGEP (Implementación de la Guía de Estilo en PHP), cuya primera versión estable se etiquetó el 16/06/2004.
Cuando el 16/11/2006 fue liberado con licencia GNU GPLv2 versión 2 tomó la denominación de gvHidra. Otros hitos importantes en la historia del proyecto son la publicación de la versión 2.0 compatible con PHP5 el 22/10/2007.

gvHidra se expande a distintas Consellerias (antes de surgir la DGTI) y también a otras organizaciones. Por ejemplo, hay aplicaciones desarrolladas con gvHidra en el grupo Johnson Controls, en el hospital "Lluís Alcanyís" de Xàtiva, en el ayuntamiento de Borriana o en varias PYMES de la Comunidad Valenciana.

La última versión estable es la v4.2.8 publicada el 15/10/2015, se esta preparando la versión 4.2.9 cuyo lanzamiento será en Diciembre y se está trabajando ya en la versión 5.0 (que contendrá cambios sustanciales) prevista para el 2016.

3.2 Comunidad

La información completa al respecto de la comunidad gvHidra puede encontrarse en www.gvhidra.org.

Para un desarrollador, los contenidos más relevantes serán el manual de usuario así como el espacio de comunicación con otros desarrolladores (lista de correo gvHidra).

Otros elementos no menos importantes son la forja del proyecto, donde podremos seguir su hoja de ruto o *roadmap*, conocer sus nuevas funcionalidades o *features* así como estar al tanto de os posibles *bugs*. En el proyecto están dado de altas distintos colaborados (*commiters*) y el código fuente está disponible de forma pública.

3.3 Proyectos satélites

En el desarrollo de aplicaciones con gvHidra suele ser recurrente la incorporación de otros proyectos, que si bien no forman parte del core, son fundamentales en el desarrollo de aplicaciones con este entorno.
Trataremos superficialmente los principales desde le punto de vista de la DGTI.

3.3.1 Jasper

El proyecto Jasper surge tras constatar las necesidades de contar con una herramienta de reporte o generación de informes de uso sencillo. Las soluciones que se habían utilizado hasta entonces se basaban en XSL-FO y requerían mucho conocimiento del dicho lenguaje por parte de los que realizaban los informes. Tras estudiar distintas posibilidades (OpenRpt, Agata, jFreeReport ...), nos inclinamos por la pareja formada por JasperReports e iReport (https://community.jaspersoft.com/), proyectos que han superado con creces las expectativas iniciales.

El proyecto Jasper es una librería o SW híbrido entre Java y PHP que permite utilizar estas herramientas dentro de gvHidra o cualquier otro proyecto PHP comunicando ambas plataformas a través de XML.

En las aplicaciones gvHidra que lo utilizan (la gran mayoría) aparece dentro de la carpeta “include” a nivel de aplicación.

3.3.2 WSComun

Este proyecto no es más que una librería cliente de los WS-Secure de PAI (eSIRCA), es decir, permite consumir los servicios web que se ofrecen en la Plataforma Autonómica de Intermediación de Datos Segura (PAI), dentro del proyecto e-SIRCA. de la DGTI.
Las aplicaciones gvHidra (o PHP) que necesiten esta librería podrán incluir este módulo y pasar a

utilizar de forma sencilla SAFE (autenticación LDAP), el modulo de firma, el gestor documental GDE, etc...

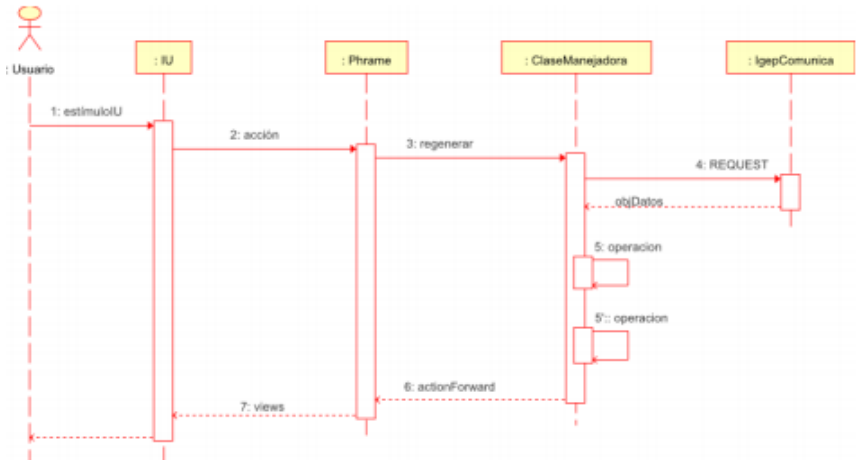
3.3.3 Genaro

Es un generador de código de aplicaciones gvHIDRA. Mediante una interfaz web, permite crear de forma rápida formularios funcionales de mantenimientos CRUD (búsqueda, alta, baja y modificación).
Para ello, únicamente requiere de la especificación de la conexión a la BD (a la que se interroga por ingeniería inversa) la tabla o tablas de la BBDD que van a atacarse y el patrón de interfaz de usuario elegido.

La mejora que se pretende obtener con esta herramienta es tanto académica[5] (bajar el umbral de aprendizaje del *framework*) como de mejora de productividad en pantallas o casos de usuario básicos.

3.4 Características funcionales

- Arquitectura MVC: El *framework* garantiza la arquitectura MVC forzando la separación de la lógica de negocio de la presentación mediante una distribución física de los ficheros fija.
  - Modelo: El FW provee de clases base de donde heredar apoyadas en distintas librerías para la persistencia de los datos transparente al SGBD, MAIL para el envío de correos, extensiones SOAP para los Web Services, etc.
  - Vista: Para la implementación de la vista se utiliza el motor de plantillas Smarty, sobrecargando el mismo con una estructura de *plugins* jerárquica y diversas librerías JS (menús dinámicos, máscaras, diálogos modales, etc.)
  - Controlador: Como controlador se adopto como base el proyecto Phrame (basado en Apache java Struts). Al morir el proyecto GPL, se incorporó al core y se hizo un *fork* del mismo, añadiendo funcionalidad extra (recargas sin refresco de pantalla, etc.)

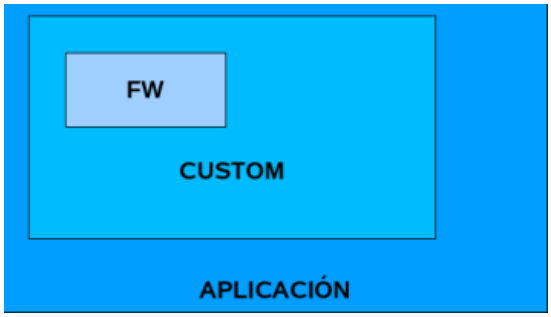


- Patrones: para facilitar el desarrollo se han definido una serie de patrones de comportamiento de interfaz. Estos patrones (los veremos con más detalle más adelante) definen la forma de representación de la información (formato tabular, registro, etc) y la forma con la que interacciona el usuario con dicha información. Esto nos lleva a que con la selección de un patrón estamos marcando el diseño global de la ventana.
- Componentes o artefactos de interfaz complejos: la experiencia acumulada nos dice que todas las aplicaciones necesitan de una serie de componentes "complejos". Ventanas de selección, listas enlazadas, acciones de interfaz (en web tecnología AJAX), mensajes de información, etc. El *framework* genera estos componentes simplificando su utilización en las aplicaciones.
- Operaciones preprogramadas y parametrizables: al igual que con los componentes, cierta

[5] La primera versión de "Genaro" es un proyecto final de carrera de un alumno de informática en la UPV

problemática se repite en todas las aplicaciones. Por esa razón, se ha generalizado y resuelto en el framework; siendo incorporada a las aplicaciones de forma transparente. Algunos de los casos más típicos son: control de concurrencia, búsquedas inteligentes, CRUD, validaciones, transacciones, etc.

- Soporte a diferentes SGBD: a través del proyecto PEAR::MDB2 (y PDO) el *framework* permite trabajar con diversos SGBD (Oracle, Postgresql, SQL Server, MySQL, MariaDB, SQLite)
- *Customs*, temas o pieles: para poder definir las particularidades propias de la organización, el *framework* se distribuye en una arquitectura App/Custom/core que permite modificar tanto el aspecto (CSS, imágenes, etc) como definir extensiones en el comportamiento propio para la organización.



- Autentificación: Para poder incorporarse en diferentes organismos, el framework incorpora un mecanismo de validación extensible siendo capaz de acoplarse a cualquier sistema de validación a través de una API basada en PEAR:Auth. Actualmente, las aplicaciones gvHidra se validan contra el módulo SAFE, contra CMS como DRUPAL, con certificados de la ACCV, contra módulos de las distintas *Intranets* corporativas basadas en *user/pass*, etc.
- Depuración y Auditoría: Tanto para facilitar la depuración durante el desarrollo como para realizar tareas de auditoría en explotación, el *framework* incorpora un mecanismo sencillo mediante registro en BD.

3.5 Guía de estilo

Como ya hemos comentado, una de las características de gvHidra es que su interfaz se basa en una guía de estilo definida de forma previa al framework. A pesar de que lo más sencillo es navegar por una aplicación gvHidra para ver y entender la guía de estilo, trataremos de realizar una pequeña introducción en este documento, por si mientras se lee el mismo no se dispone de un portátil.

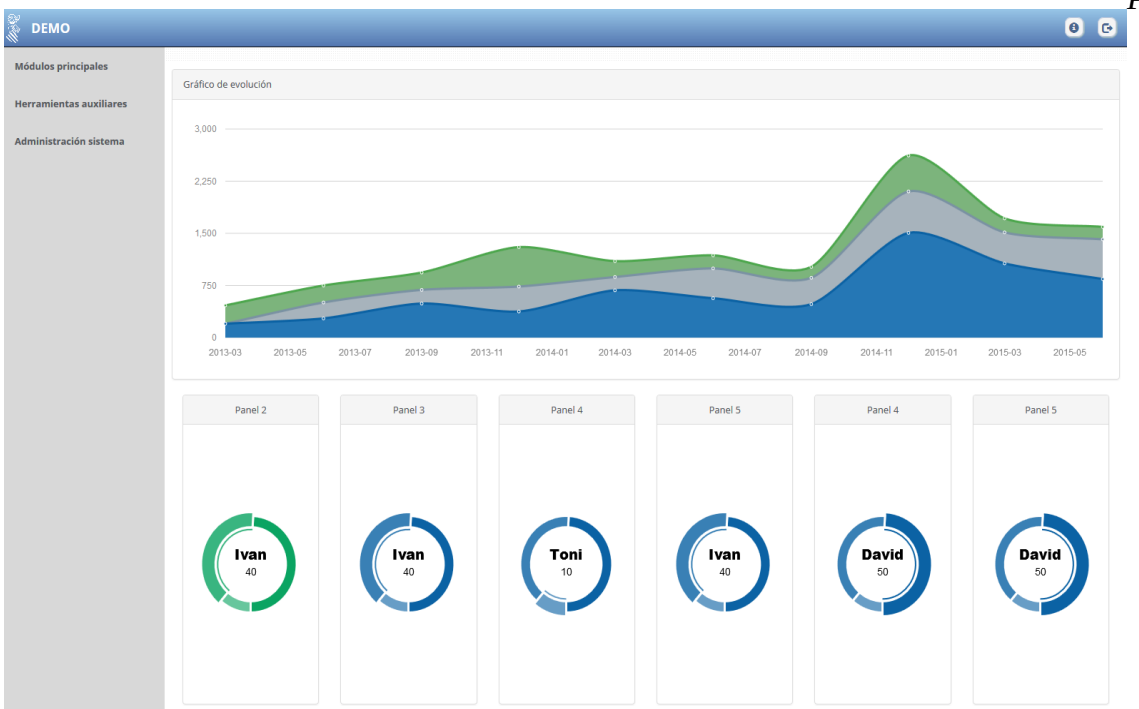
La versión que se presenta en la presente edición del curso es la actualización basada en bootstrap, que se incorporá a partir de la versión 4.2 a GvHidra, no es necesario instalar nada ya que viene **incorporado** en su **núcleo** (igep)**.** Gracias a Bootstrap las aplicaciones realizadas con gvHidra son *responsive*, o adaptaivas, por lo tanto se verán bien a diferentes resoluciones. Por ejemplo:

Resolución

normal a pantalla completa

Por otra parte gvHidra también usa componentes del *framework* como son los *Glyphicon*, ventanas emergentes, *dashboard*, gráficos...

P



A continuación haremos un repaso sobre la interfaz de las aplicaciones GvHidra.

3.5.1 Pantalla inicial

La pantalla inicial está dividida en dos zonas. Una superior que contiene el título de la aplicación, la versión de la misma y el usuario que la está utilizando, y en la inferior aparecen tres menús distribuidos en columnas. Estos menús son:

- Módulos principales: bloque que contiene los enlaces a las pantallas principales de la aplicación, estas pantallas componen el núcleo funcional del aplicativo, ya que desde ella se realizan las tareas funcionales imprescindibles para que la aplicación cumpla su finalidad. Por ejemplo: Crear una nueva actuación fiscal...
- Herramientas auxiliares: Se personalizan en función del usuario, normalmente no aparecen entradas.
- Administración del sistema: Información sobre la versión de la aplicación y la del framework gvHidra.

En el caso de que la resolución de la pantalla no permita la distribución de las tres columnas en horizontal quedaría de la siguiente forma:

3.5.2 Elementos principales

La pantalla se divide en tres áreas:

- Barra superior: Dividida en tres zonas. A la izquierda se encuentra el menú propio de la aplicación, en la zona central el título de la aplicación y versión de la misma, y por último, a la derecha de la barra se encuentra un botón *info*, que nos da más información sobre el usuario conectado y rol, y los botones que nos permiten volver a la pantalla principal y salir de la aplicación.
- Botonera: Barra en la que irán apareciendo los botones correspondientes a los paneles que estén activos y/o a los que se pueda acceder.
- Panel/es: donde se mostrará la información o el/los formulario/s correspondiente/s a la opción elegida.

3.5.2.1 Barra superior

La barra superior tiene el siguiente aspecto:

Y sus componentes son:

- Menú desplegable de la aplicación, donde aparecerán las opciones del menú elegido en la pantalla de inicio. Las opciones de este bloque (menú principal o tablas auxiliares) podrán volver a desplegarse para entrar en mayor detalle. De esta forma, evitamos tener que volver al inicio para poder acceder a otro enlace de un mismo menú.
- Título de la aplicación y versión correspondiente
- Botón *info* que mostrará una capa con la información correspondiente al usuario conectado (nombre usuario, rol...).
- Botones. Con el primero de ellos podremos volver a la pantalla principal de la aplicación, y con el segundo, saldremos de la aplicación.

3.5.2.2 Botonera

La botonera se encuentra justo debajo de la barra superior, y tiene el siguiente aspecto:

En ella aparecerán los botones correspondientes a los diferentes paneles o modos de trabajo sobre los que se puede trabajar con la información. Los modos de trabajo no son más que las distintas formas que tenemos de representar la información con la que estamos trabajando (filtro, información en formato tabular, información en formato ficha o registro...)

Los botones tendrán dos estados:

- Botón modo activo
- Botón modo inactivo

La aparición de estos botones tendrá un orden secuencial al modo de trabajo, es decir, el primer botón en aparecer cuando se entre será el de “Nueva búsqueda”, por lo tanto la botonera tendrá el siguiente aspecto:

Una vez busquemos y nos devuelva el resultado, pasando así al siguiente modo de trabajo, por ejemplo “tabular”, la botonera tendrá el siguiente aspecto:

Aquí ya vemos que ha aparecido un segundo botón a la derecha del de “Nueva búsqueda”, “Listado”. “Listado” corresponde al panel que ahora tendríamos activo, el botón aparece en modo inactivo, en cambio el de “Nueva búsqueda” cambia y se pone en modo activo.

Si la edición de uno de los registros del panel “Listado” nos llevara a un panel registro, la botonera quedaría de la siguiente forma:

Vemos que el botón “Edición” es el que aparece en modo inactivo, y los otros dos anteriores en modo activo.

Recapitulando, el funcionamiento de la botonera tiene relación con el modo de trabajo y el orden del mismo. El último panel sobre el que se trabaja siempre tendrá su botón más a la derecha y estará en modo inactivo, los botones a la izquierda de él estarán activos y nos permitirán volver a activar dichos paneles. Cuando pulsemos sobre uno de los botones activos se desplegará el panel correspondiente encima del panel activo, si volvemos a pulsar sobre el botón, porque no vamos a realizar nada en ese panel, se ocultará.

3.5.2.3 Paneles

Zona donde encontraremos el formulario correspondiente a la opción elegida para poder trabajar con él. El panel está dividido en cuatro zonas diferenciadas:

- Barra superior del panel.

Encontramos las siguientes zonas:

- ➢ Título de la pantalla en la que nos encontramos.
- ➢ Si los hay, aparecerán botones que actuarán sobre el formulario que aparece en el panel (concretamente sobre las líneas seleccionadas si es un panel tabular, o sobre el registro visible si nos encontramos en un panel ficha) y no efectuarán acciones sobre la base de datos.

- Área de trabajo.

Zona donde se presentará la información dependiendo del modo de trabajo en el que nos encontremos. Es la zona intermedia entre la barra superior y la barra pie.

- Barra pie

Encontramos las siguientes zonas:

- ➢ Paginador, nos permite navegar por los resultados.
- ➢ Botones destinados a ejecutar acciones, desde guardar datos en la base de datos, realizar algún cálculo, o cancelar la acción actual.

3.5.3 Ventanas de aviso, alerta y error

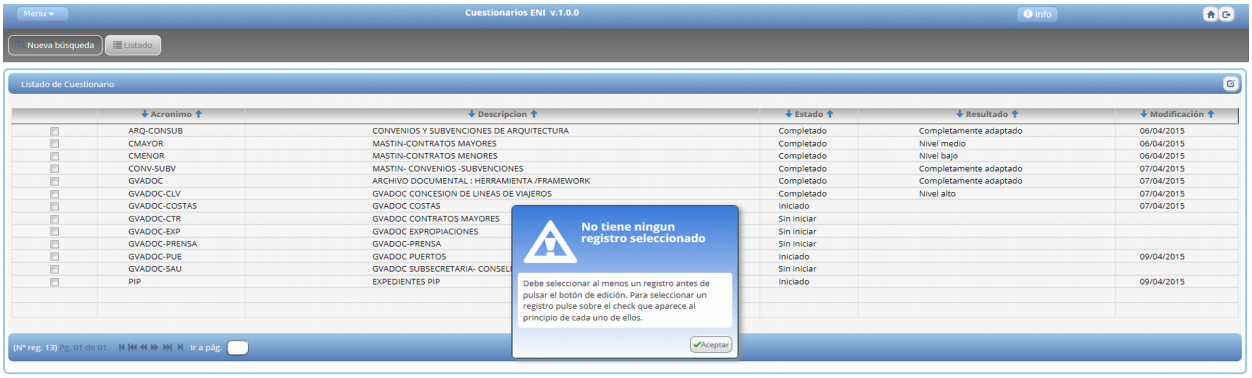
Muestran información importante para el transcurso de la aplicación. Cuando una ventana de este tipo salta al ejecutar cualquier acción, al cerrarlas volveremos al panel donde nos encontrábamos.

Tiene un tamaño más pequeño que el panel y no nos deja ejecutar ninguna acción hasta haberlas cerrados, tienen un funcionamiento modal.

El color del aviso nos indica distintos tipos de mensajes, siendo estos:

- Rojo: errores ocurridos, no deja continuar hasta que se corrija el error.
- Amarillo: alertas
- Azul: avisos u otro tipo de información que se quiere dar al usuario.

Ejemplo de aviso:



3.5.4 Elementos de los paneles

La función de los paneles es organizar la información en la zona de trabajo. Intentaremos agrupar esta información como conjunto homogéneo de datos, por eso aparecerán tantos paneles como grupos de información.

3.5.4.1 Botones tooltip

Este tipo de botones representan aquellas acciones o utilidades que trabajan sobre un panel a nivel de pantalla, y raramente sobre la Base de Datos de forma directa.

Suelen estar ubicados en dos lugares distintos:

- Los encontramos en la cabecera del panel, situados a la derecha. Estos actúan sobre el panel y lo predisponen para alguna acción. Ejemplos: Modificar, Insertar, Limpiar, …
- Los que encontramos repartidos por el panel y se utilizan para completar información de algún campo. Aparecen situados a la derecha del campo al que afectan. Ejemplos: Ventana selección, Calendario…

Veamos una pequeña descripción de funcionamiento de los botones ToolTip más comunes:

- Nuevo: Crea un nuevo registro. En el caso de un mantenimiento tipo ‘tabla’ pulsar este botón indica que se activen las filas restantes para finalizar la tabla o las filas de la página siguiente en el caso de estar completa la última. Si el mantenimiento es tipo ‘ficha’ aparecerá una ficha en blanco donde introduciremos los valores del registro a insertar.
- Modificar: Activa los campos para poder modificar un registro.
- Eliminar: En el caso de mantenimiento tipo tabular (tabla) se marcarán como eliminados los registros seleccionados, estos se seleccionan activando el checkbox que se encuentra al principio de la fila. Y en el caso de mantenimiento tipo registro (ficha individual) el registro donde nos encontremos en el momento de pulsar el botón es el que pasará a estado borrado.
- Restaurar: Este botón ToolTip tiene dos acciones: Refrescar o vaciar. En el caso de vaciar, el pulsarlo supone que todos los campos del registro aparezcan vacíos. Y refrescar supone que aparezcan los valores del registro anteriores a la modificación.

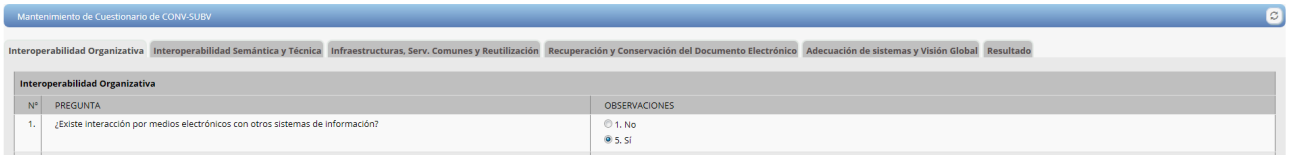
3.5.4.2 Botones

Estos botones son los que aparecen en la barra inferior del panel. Estos botones son los que normalmente llevan a cabo las acciones sobre la base de datos.

En estos botones aparecerá un texto que deberá ser lo más significativo posible con relación a la acción a ejecutar.

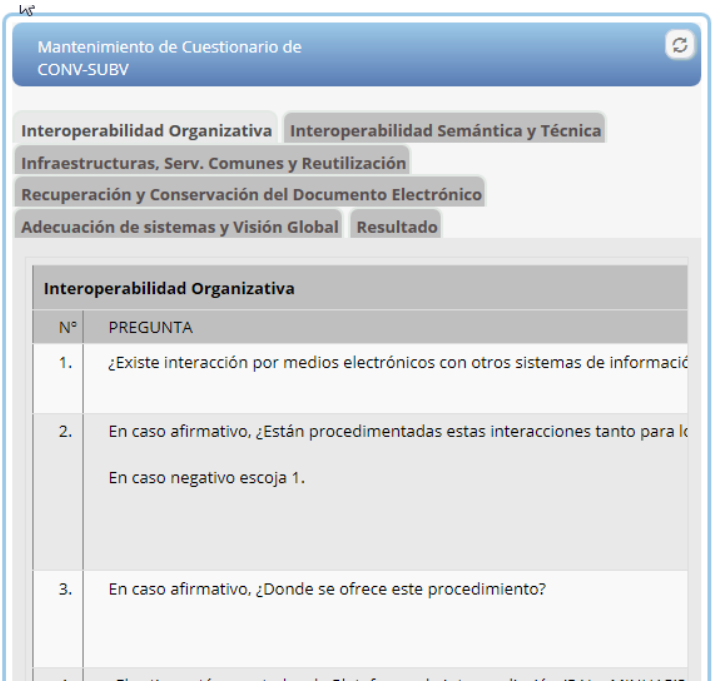
3.5.4.3 Solapas agrupadoras

Este elemento se utiliza para agrupar en un mismo panel información relacionada. Serán necesarias cuando haya que mostrar mucha información que implique que el panel se haga demasiado largo. El título de la pestaña nos indicará el tipo de la información que contendrá.



La solapa "*Interoperabilidad Organizativa*" de la imagen, es la que está activa, es decir, se visualizará el panel con su información, el resto de solapas están "ocultas", pulsando en ellas visualizaremos su información.

Si por la resolución de la pantalla no pudieran aparecer todas las solapas en línea, éstas se irían agrupando por filas, quedando de la siguiente forma:



3.5.4.4 Paginador

En la barra inferior del panel aparecerá un elemento paginador, que como su propio nombre indica, nos permitirá ir cambiando de página. Si estamos trabajando con un panel "tabular" cada página que avancemos implicará desplazarse varios registros. En cambio, si trabajos en modo registro, cada pulsación o cambio de página, implica el avance unitario de registro a registro sobre el conjunto total

En el caso de un panel ficha y cuando la información a mostrar corresponda a más de un registro,

con el paginador podremos ir moviéndonos por los distintos registros. En este caso, cuando se modifique algún dato de algún registro, aparecerá una lista desplegable en la que se irán acumulando las páginas donde hay cambios para informar al usuario antes de que grabe en la BD. El registro que ha sido modificado también incorpora al lado del paginador una imagen de alerta.

En el caso de un panel de tipo tabla, el paginador es útil cuando el número de registros a mostrar es superior al número de filas que aparecen en pantalla, por lo que se generarán más páginas con el resto de registros.

3.6 Patrones de interfaz

La literatura de investigación sobre patrones de interfaz en el mundo de la informática ha sido siempre fecunda, gvHidra no trata de ser un paradigma de ingeniería del SW en este sentido. Cuando en gvHidra nos referimos a patrones de pantalla, es porque ha medida que se han ido realizando aplicaciones con el FW se han ido detectando casos recurrentes en el funcionamiento de la lógica de la interfaz de usuario.

En base a esta experiencia, en gvHidra se da soporte a ciertos patrones de pantalla permitiendo que el desarrollo de ciertas pantallas o formularios sea mucho más rápido cuando nos acoplamos a uno de estos patrones.

Al analizar una aplicación podemos plantear un formulario o ventana de muchas formas, casi todas todas son posibles de implementar, la diferencia está en el esfuerzo requerido para ello. Si conseguimos adaptarnos a los patrones de pantalla/formulario existentes, la productividad será máxima, sin embargo, a medida que nuestros diseños de pantallas y formularios se alejen de estos patrones, la productividad será mucho más baja, pues el FW no dará tanto soporte al programador y este tendrá que programar más para obtener el comportamiento deseado.

El conseguir la mejor aproximación para un diseño de interfaz que satisfaga al usuario final y no representen merma en la productividad atiende a diversos factores. Vamos a ofrecer una aproximación basada en las relaciones y su cardinalidad (E/R, UML o el diagrama lógico de la BD)
En cualquier caso, debe considerarse orientativa, dado que la experiencia en el uso del FW y la buena captación de los requisitos funcionales en la fase de análisis serán con seguridad mucho más determinantes que las pautas de este documento.

Los patrones principales están clasificados por un código que trata de reflejar sus características:

Pi[Mj]([???]j)

donde:

Pi es el número de paneles del patrón. El valor de i es 1 o 2.

Mj([???]). Es el número de modos posibles del panel. El valor de j es 1, 2 o 3 y entre paréntesis se especifica el tipo de cada modo. FIL (Filtro), LIS (listado tabula) o EDI (Ficha).

Atendiendo a esta clasificación, gvHidra asiste con multitud de automatismos los patrones siguientes.

3.6.1 Patrones simples

Se corresponden con patrones donde sólo se presenta un panel (con uno o varios modos de trabajo o pestañas). Su uso habitual es el acceso (tanto de lectura como de modificación) a una sola “entidad de datos” (tabla, clase, pojo, WSService...) o a entidades relacionadas con cardinalidad 1 a 1, aunque su uso puede ser otro (ser simplemente una interfaz de usuario para lanzar otras acciones, alimentar y/o leer desde una varias entidades de datos y facturarlas en un solo formulario a nivel de interfaz...)

3.6.1.1 P1M2(FIL-LIS) o patrón de listado tabular

Se compone de un modo panel o pestaña de filtro (búsqueda) y un panel (pestaña) en formato tabular donde podemos:

- Ver los resultados obtenidos al ejecutar el filtro definido en FIL.
- Insertar/Borrar o modificar algún resultado (CRUD) en el panel LIS.

3.6.1.2 P1M1(FIL) o patrón de filtro, búsqueda o lanzamiento

Se compone solamente de una pestaña o modo panel de filtro. Nos permite:

- Lanzar una acción de la que no se espera resultado con iteración compleja en la interfaz:
  - Lanzar un reporte o listado (Jasper, OpenDocument...)
  - Lanzar un proceso (Mailing, Batch...)
  - Servir de inicio a un salto a otro patrón de interfaz

3.6.1.3 P1M1(LIS) o Patrón Tabular sin búsqueda

Se compone solamente de un modo panel tabular, donde se presenta información directamente, bien al acceder a la pantalla desde el menú o bien llegando a ella desde un patrón distinto donde se ha realizado un salto. Nos permite:

- Ver resultados (el origen de los mismos será una búsqueda fija sin filtro, el origen de un salto...)
- Insertar/Borrar o modificar algún resultado (CRUD) en el panel LIS.

3.6.1.4 P1M1(EDI) o Patrón registro para alta Masiva

Se compone solamente de un modo panel registro, donde se presenta información directamente, bien desde accediendo desde el menú o bien llegando mediante salto desde otro patrón. Nos permite:

- Ver los resultados obtenidos al ejecutar el filtro definido en FIL.
- Insertar/Borrar o modificar algún resultado (CRUD) en el panel LIS.

Su uso más habitual es el alta continua de registros, se presenta en registro en blanco y tras pulsar guardar, se inserta en BD y se vuelve a presentar el registro en blanco.

3.6.1.5 P1M2(FIL-EDI) o Patrón registro o ficha

Se compone de un panel o pestaña de filtrado (búsqueda) y un panel (pestaña) en formato ficha o registro donde podemos:

- Ver los resultados obtenidos al ejecutar el filtro definido en FIL.
- Insertar/Borrar o modificar algún resultado (CRUD) en el panel LIS.

El funcionamiento es similar al patrón tabular, pero la visualización es más detallada, pues en lugar de trabajar en formato tabular (viendo un conjunto de registro con menos detalle) se trabaja en formato ficha, viendo todos los campos de un registro.

3.6.1.6 P1M1(EDI) o Patrón registro o ficha sin búsqueda

Se compone solamente de un panel registro o ficha, en el cual, se presenta la información directamente (o aparece en blanco para introducir la misma). Podemos llegar a este patrón, bien desde el menú o bien desde un patrón distinto donde se ha realizado un salto. Nos permite:

- Ver resultados (el origen de los mismos será una búsqueda fija sin filtro, el origen de un salto...)
- Insertar/Borrar o modificar algún resultado (CRUD) en el panel LIS.

Su uso es variado, por ejemplo, podemos emplearlo en una ventana modal para resolver el alta de claves ajenas. (Ej. Dando de alta una factura, vemos que el proveedor no existe, sin salir de la ventana de Alta, invocamos una ventana modal de patrón registro sin búsqueda, damos de alta el proveedor y cerramos la ventana modal).

3.6.1.7 P1M3(FIL-LIS-EDI) o patrón tabular-registro

Es el patrón por excelencia en el mantenimiento de datos. Compuesto de tres paneles o pestañas, el panel o pestaña de filtrado (búsqueda), un panel tabular y un panel en formato ficha o registro. El funcionamiento habitual, es usar el panel filtro para obtener un conjunto de registros, que se visualizan en el panel tabular para (comportamiento habitual):

- Borrar registros en el formato tabular
- Insertar y/o editar los datos en el panel registro.

Permite hacer alguna variación (por ejemplo, borrar también en el panel registro, o realizar el alta o la edición en el panel tabular, aunque sólo tiene sentido en casos muy concretos.

El funcionamiento es similar al patrón tabular, pero la visualización es más detallada, pues en lugar de trabajar en formato tabular (viendo un conjunto de registro con menos detalle) se trabaja en formato ficha, viendo todos los campos de un registro.

3.6.2 Patrones complejos

Son patrones de interfaz que proponen una interfaz para interactuar con más de una entidad de datos (clase, tabla..) y las relaciones (con distinta cardinalidad) entre las mismas.
En modelos grandes y complejos, la traducción de estas relaciones a la interfaz de usuario puede traer problemas de complejidad en la implementación (interfaz muy compleja y de difícil mantenimiento) y de usabilidad (al usuario no le resulta intuitiva la interfaz). Por lo que en esos casos, y a nivel de interfaz, resulta mucho más productivo y sencillo, descomponer esas relaciones y optar por componer la solución con más de una pantalla (patrón), enlazarlos con saltos y alimentar la relación de datos de forma correcta, sin complicar la interfaz.

Los patrones a los que el FW presta asistencia automática o semiautomática son los Maestros-Detalle, que se utilizan habitualmente para resolver relaciones entre entidades de datos con cardinalidad [1,*] o resolver un extremo de navegación entre entidades de datos que presentan cardinalidad [*, *]. Y el patrón árbol, que es el más versátil, aunque es más complejo de manejar.

El general la aplicación de las distintas variaciones de los patrones Maestro Detalle depende principalmente de la cantidad de información a mantener en cada una de las entidades de datos relacionadas, y de cómo creamos que le es más ágil trabajar al usuario.

Pueden combinarse con saltos y/o ventanas modales para completar aspectos funcionales de la interfaz o preferencias concretas del usuario.

3.6.2.1 P2M2(FIL-LIS)M1(LIS) Patrón Maestro Tabular - Detalle Tabular

Se compone de:

- Un panel Maestro dispuesto en la parte superior de la pantalla con:
  - Una pestaña o modo de trabajo (modo panel) de filtro o búsqueda
  - Una pestaña o modo de trabajo tabular que presenta los datos en formato listado.
- Un panel Detalle dispuesto en la parte inferior de la pantalla con:
  - Una pestaña o modo de trabajo tabular que presenta los datos relacionados con el registro maestro seleccionado como un listado.

3.6.2.2 P2M2(FIL-LIS)M1(EDI) Patrón Maestro Tabular - Detalle registro

Se compone de:

- Un panel Maestro dispuesto en la parte superior de la pantalla con:
  - Una pestaña o modo de trabajo (modo panel) de filtro o búsqueda
  - Una pestaña o modo de trabajo tabular que presenta los datos en formato listado.

- Un panel Detalle dispuesto en la parte inferior de la pantalla con:
    - Una pestaña o modo de trabajo registro que presenta los datos relacionados con el registro maestro seleccionado como una colección de fichas.

3.6.2.3 P2M2(FIL-EDI)M1(EDI) Patrón Maestro Registro - Detalle Registro

Se compone de:

- Un panel Maestro dispuesto en la parte superior de la pantalla con:
    - Una pestaña o modo de trabajo (modo panel) de filtro o búsqueda
    - Una pestaña o modo de trabajo registro donde se presentan los datos en modo ficha.
- Un panel Detalle dispuesto en la parte inferior de la pantalla con:
    - Una pestaña o modo de trabajo registro que presenta los datos relacionados con el registro maestro activo.

3.6.2.4 P2M2(FIL-EDI)M1(LIS) Patrón Maestro Registro - Detalle Tabular

Se compone de:

- Un panel Maestro dispuesto en la parte superior de la pantalla con:
    - Una pestaña o modo de trabajo (modo panel) de filtro o búsqueda
    - Una pestaña o modo de trabajo registro donde se presentan los datos en modo ficha.
- Un panel Detalle dispuesto en la parte inferior de la pantalla con:
    - Una pestaña o modo de trabajo registro que presenta los datos relacionados con el registro maestro activo.

3.6.2.5 P2M2(FIL-EDI)M1(LIS-EDI) Patrón MD Maestro Registro - Detalle Tabular-Registro

Se compone de:

- Un panel Maestro dispuesto en la parte superior de la pantalla con:
    - Una pestaña o modo de trabajo (modo panel) de filtro o búsqueda
    - Una pestaña o modo de trabajo registro donde se presentan los datos en modo ficha.
- Un panel Detalle dispuesto en la parte inferior de la pantalla con:
    - Una pestaña o modo de trabajo tabular que presenta los datos relacionados con el registro maestro activo como un listado.
    - Una pestaña o modo de trabajo registro que presenta y permite actualizar y/o insertar los datos de un detalle concreto seleccionado en el modo de trabajo listado del panel detalle.

3.6.2.6 Patrón Maestro N-Detalles

Se compone de:

- Un panel Maestro dispuesto en la parte superior de la pantalla que puede cumplir con cualquier combinación de los paneles maestros anteriores.
- Tantos paneles detalle como deseemos, cada uno de ellos (individualmente) puede corresponderse a un patrón detalle distinto de los anteriormente expuestos.

3.6.2.7 Patrón Árbol

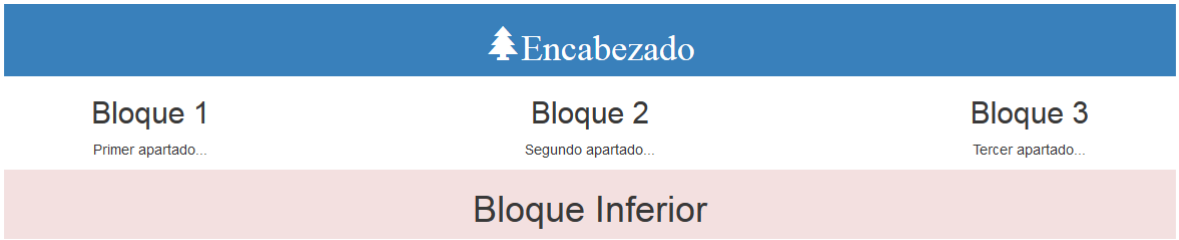
Patrón muy versátil que permite mantener entidades relacionadas de forma jerárquica. Se encuentra en el límite de la productividad (es un patrón complejo de implementar y mantener) y la usabilidad (si el modelo de datos no tiene una clara relación jerárquica para el usuario, puede perderse).

4 Casos prácticos

4.1 Caso práctico: Introducción a PHP

4.2 Caso práctico: Conceptos básicos Bootstrap

4.2.1 Solución



Pasos a seguir:

1-Crear un archivo **.txt** y cambiarle la extensión a **.html.**

2-Copiar la siguiente plantilla a nuestro documento:

```
<!DOCTYPE html>
<html lang="es">
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
<head>
  <title>Introducción a Bootstrap</title>
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="stylesheet"
href="http://maxcdn.bootstrapcdn.com/bootstrap/3.3.5/css/bootstrap.min.css">
</head>
<body>
  <div class="container text-center">
     <div class="row bg-primary">
        <h1><span aria-hidden="true" class="glyphicon glyphicon-tree-
conifer">Encabezado</span></h1>
     </div>

   <div class="row">
    <div class="col-md-3">
      <h2>Bloque 1</h2>
      <p>Primer apartado...</p>
    </div>
    <div class="col-md-6">
      <h2>Bloque 2</h2>
       <p>Segundo apartado...</p>
    </div>
    <div class="col-md-3">
      <h2>Bloque 3</h2>
       <p>Tercer apartado...</p>
    </div>
  </div>

    <div class="row bg-danger">
       <h1> Bloque Inferior </h1>
    </div>
</div>
```

```
</body>
</html>
```

Clases de Boostrap a utilizar:

- **container**: contenedor principal de la pantalla.

Se define:

- **row**: crear un contenedor.

Se define:

- **col-sm-**x: crear columnas dentro de un row.

Se define:

Reemplazar la x por el número que se quiera usar para estructurar la pantalla.

- **text-center:** centrar texto.

Se define: class="text-center"

- **bg-primary, bg-danger**: color de fondo del div(contenedor).

Se define: class="bg-primary" y class="bg-danger".

- **Glyphicon-tree-conifer:** inserta el Glyphicon del árbol.

Se define:

Se han utilizado utilizado etiquetas <h1>, <h2>… <p> para encapsular el texto.

IMPORTANTE: Si se quiere crear un contenedor que ocupe las 12 columnas, no es necesario usar "**col-sm-12",** basta con dejar únicamente la etiqueta row y automáticamente se le asignará todo el espacio de las columnas.

4.3 Caso práctico: Ejecución de DEMO-GVHIDRA

Para ejecutar la aplicación DEMO-GVHIDRA en local, necesitamos crear la estructura de la BD y luego importar los datos. Creación de la BD:

```
/* Creamos la BD */
CREATE DATABASE demo
  WITH OWNER = postgres
       ENCODING = 'UTF8'
       TABLESPACE = pg_default
       CONNECTION LIMIT = -1;

/* Creamos el Schema */
CREATE SCHEMA demo
  AUTHORIZATION postgres;
COMMENT ON SCHEMA demo
  IS 'DEMO GVHIDRA IVAP';


/* Creación del usuario propietario */
CREATE USER demo PASSWORD 'demo';
ALTER USER demo VALID UNTIL 'infinity';

CREATE USER demo_ae PASSWORD 'demo_ae';
CREATE USER demo_ac PASSWORD 'demo_ac';
ALTER USER demo_ae VALID UNTIL 'infinity';
ALTER USER demo_ac VALID UNTIL 'infinity';

/* Creación de los roles */
CREATE ROLE rdemo ADMIN demo;
CREATE ROLE rdemo_c ADMIN demo;

/* Asignar los roles a los usuarios */
GRANT rdemo   TO demo;
GRANT rdemo   TO demo_ae;
GRANT rdemo_c TO demo_ac;

/* Permisos */
GRANT ALL ON SCHEMA demo TO postgres;
GRANT ALL ON SCHEMA demo TO demo WITH GRANT OPTION;
GRANT USAGE ON SCHEMA demo TO rdemo;
GRANT USAGE ON SCHEMA demo TO rdemo_c;

/* Dar permisos al esquema para el usuario propietario y los roles */
GRANT ALL  ON SCHEMA demo TO demo WITH GRANT OPTION;
GRANT USAGE ON SCHEMA demo TO rdemo;
GRANT USAGE ON SCHEMA demo TO rdemo_c;

/* Asignar los search_path necesarios para los usuarios */
ALTER USER demo    SET search_path = 'demo';
ALTER USER demo_ae SET search_path = 'demo';
ALTER USER demo_ac SET search_path = 'demo';

/* Opcional: Creación de infraestructura de LOG */
CREATE TABLE demo.tcmn_errlog (
        iderror numeric(9,0) NOT NULL,
        aplicacion character varying(30) NOT NULL,
        modulo character varying(10),
        version character varying(10) NOT NULL,
        usuario character varying(30) NOT NULL,
        fecha timestamp without time zone NOT NULL,
        tipo numeric(2,0) NOT NULL,
        mensaje text NOT NULL
);
ALTER TABLE demo.tcmn_errlog OWNER TO demo;
GRANT ALL ON TABLE demo.tcmn_errlog TO rdemo;

CREATE SEQUENCE demo.scmn_id_errlog
        START WITH 1
```

```
        INCREMENT BY 1
        NO MINVALUE
        NO MAXVALUE
        CACHE 1;
ALTER TABLE demo.scmn_id_errlog OWNER TO demo;
GRANT ALL ON SEQUENCE demo.scmn_id_errlog TO rdemo;
```

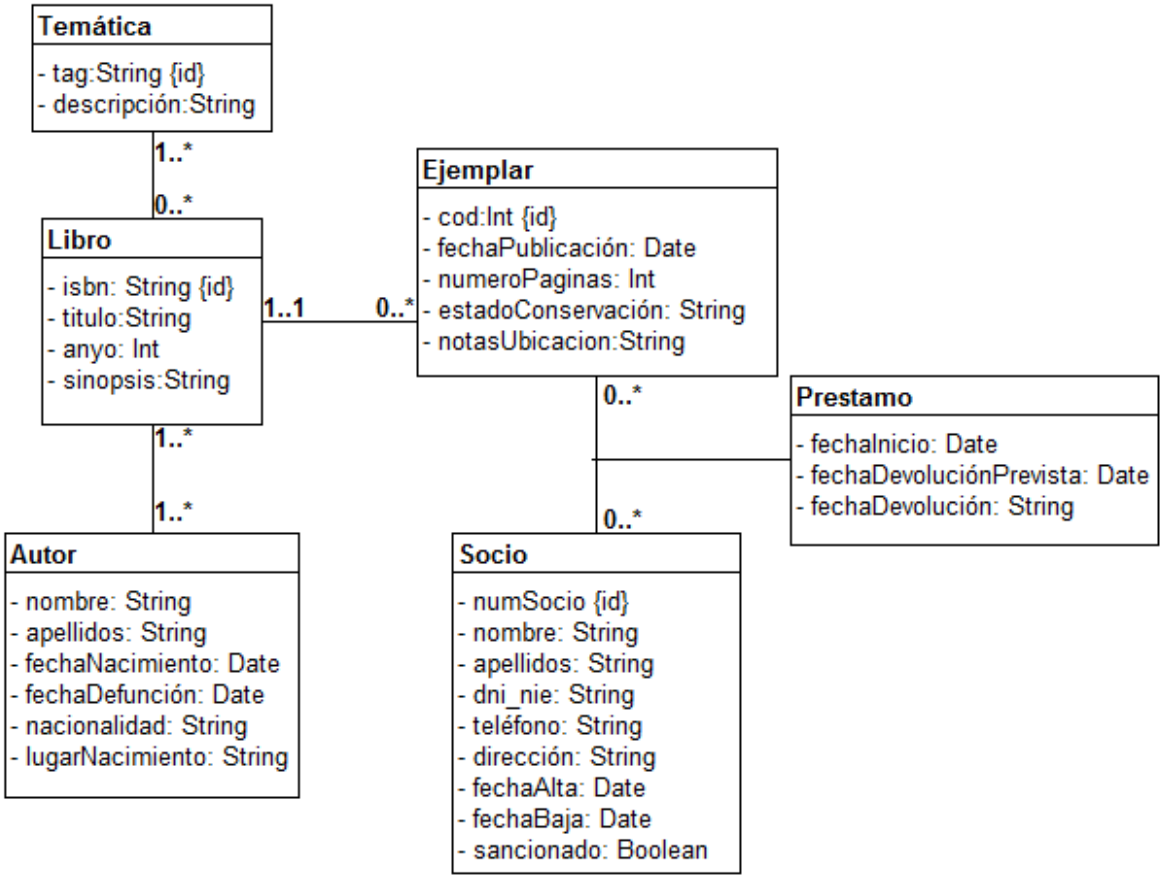
Después importamos el DUMP de datos.

4.4 Caso práctico: Patrones de interfaz gvHidra

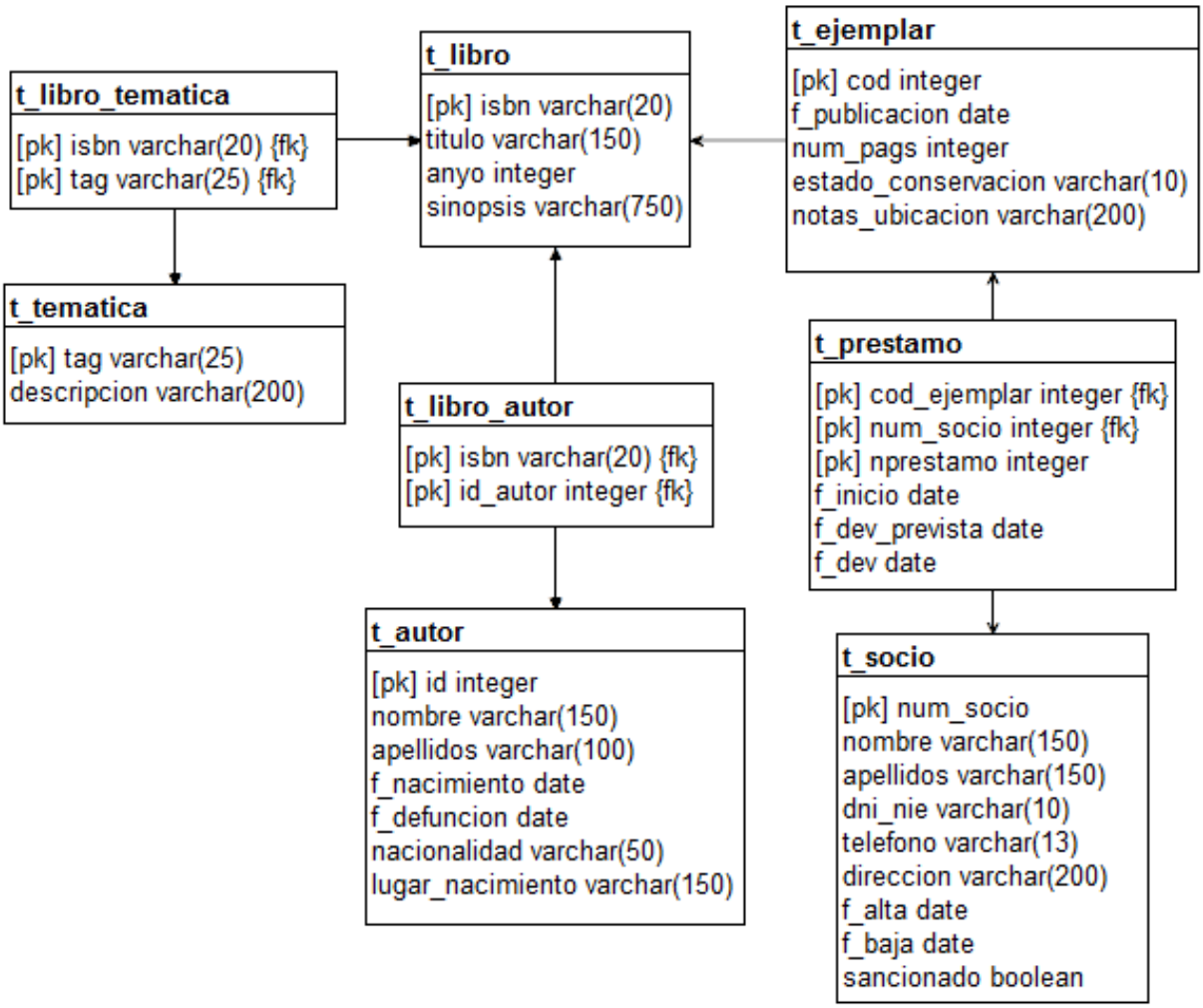
Tras introducir la interfaz, vamos a proponer soluciones o posibles patrones a aplicar ante situaciones concretas en los modelos de datos (UML y/o relacionales), que suelen ser recurrentes.

La intención es mostrar las mejores soluciones manteniendo un ratio de productividad óptimo.

Dado este sencillo modelo de datos UML sobre una biblioteca, vamos a comentar cuáles serían las soluciones más productivas al programar una aplicación gvHidra que lo alimente.



Supongamos que la traducción relacional queda así:



A continuación presentamos distintas posibilidades de diseño de interfaz, todas ellas cubrirían las necesidades de la aplicación la aplicación de forma completa y por patrones gvHidra:

Menú de la aplicación:

- Mantenimiento de temática: Patrón P1M2(FIL-LIS) o patrón tabular

- Mantenimiento de autores (alternativas):
  - Patrón P1M2(FIL-EDI) o patrón registro o ficha
  - P1M3(FIL-LIS-EDI) o patrón tabular registro

- Mantenimiento de libros o títulos (alternativas):
  - Patrón Maestro 2 Detalles. El Maestro será un P2M2(FIL-EDI) o Maestro Registro donde se informarán los datos de un libro.
    - El detalle 1 (Autores) tendrá un formato tabular y se mostrarán los autores del libro.
    - El detalle 2 (Ejemplares) El primer detalle será Detalle Tabular-Registro. En el detalle se verán los ejemplares asociados a ese título (si es que existen en la biblioteca). En el alta se dan de alta títulos y pueden darse de alta (o no) ejemplares.
  - Patrón P2M2(FIL-EDI)M1(LIS-EDI) o Maestro Registro, Detalle Tabular-Registro.
    - El maestro mantendrá los datos de un libro (título), la relación de 1..N con autores se soluciona con una lista múltiple y una ventana modal que permite

seleccionar un nuevo autor para alimentar la lista, o bien dar de alta el mismo si no existe. Si esta fuese la opción elegida, serviría la misma ventana de mantenimiento de autores, pero sería conveniente usar un patrón , en lugar del propuesto antes.

- El detalle 1 (Autores) tendrá un formato tabular y se mostrarán los autores del libro.
- El detalle 2 (Ejemplares) El primer detalle será Detalle Tabular-Registro. En el detalle se verán los ejemplares asociados a ese título (si es que existen en la biblioteca). En el alta se dan de alta títulos y pueden darse de alta (o no) ejemplares.

- Mantenimiento socio (alternativas)
  - Patrón P1M2(FIL-EDI) o patrón registro o ficha.
  - P2M2(FIL-EDI)M1(LIS) o Patrón Maestro Registro - Detalle Tabular: En el detalle se mostrará un listado de los préstamos (fecha inicio y fin y ejemplar). Desde esa pantalla se podrían también dar de alta préstamos para ese socio. El botón de alta en el detalle sería la forma de dar de alta nuevo préstamos. (ejemplar, fecha inicio y fecha devolución).

- Nuevo préstamo: P1M1(EDI) o Patrón registro para alta Masiva.Se introduce el número de socio y el número de ejemplar. Se muestra un mensaje informativo diciendo que se registró el préstamo o informando de que no puede realizarse y la causa (máximo número de ejemplares prestados, socio sancionado, etc...)

- Registrar devolución: P1M1(EDI) o Patrón registro para alta Masiva. Simplemente se introduce el código del ejemplar devuelto y se muestra un mensaje diciendo si todo está correcto o hay una sanción por devolución tardía.

Como vemos la aplicación puede cubrirse con los patrones gvHidra predefinidos y manteniendo por tanto una buena relación de productividad y asegurando que los mantenimientos posteriores son sencillos.

4.5 Caso Práctico: Embarcaciones

La intención del caso práctico es tener una primera aproximación de programación al FW gvHidra, partiremos de un enunciado algo académico, sobre el cual trabajaremos con la intención de desarrollar un prototipo de aplicación que permita su mantenimiento.

4.5.1 Enunciado

Desde la D. General de Puertos y Costas se ha realizado una petición a la DGTI para el desarrollo de una aplicación que registre información sobre las embarcaciones que hacen escala en los puertos de la C.V.

Tras una reunión con los responsables de marco de la DGTI (4h) y dos reuniones con los usuarios (8h), se prepara la documentación para comenzar a decidir sobre el proyecto:

- 3 actas de reunión (4h)
- Acta de arranque (1h)
- Toma de requisitos inicial (5h)
- Análisis del coste beneficio (4h + 3h tras cambios propuestos por calidad)

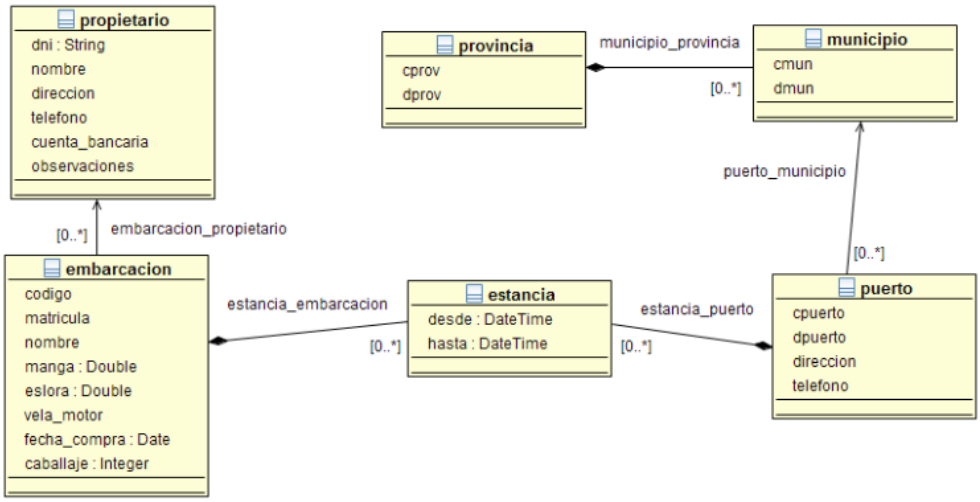
Tras una nueva reunión con los responsable de la DGTI (2h) se decide acometer el proyecto y concretar
algunos matices:

- Acta de reunión (1h)
- Se actualiza el documento de toma de requisitos (2h)
- Se redacta el documento de validación del análisis coste beneficio (3h)

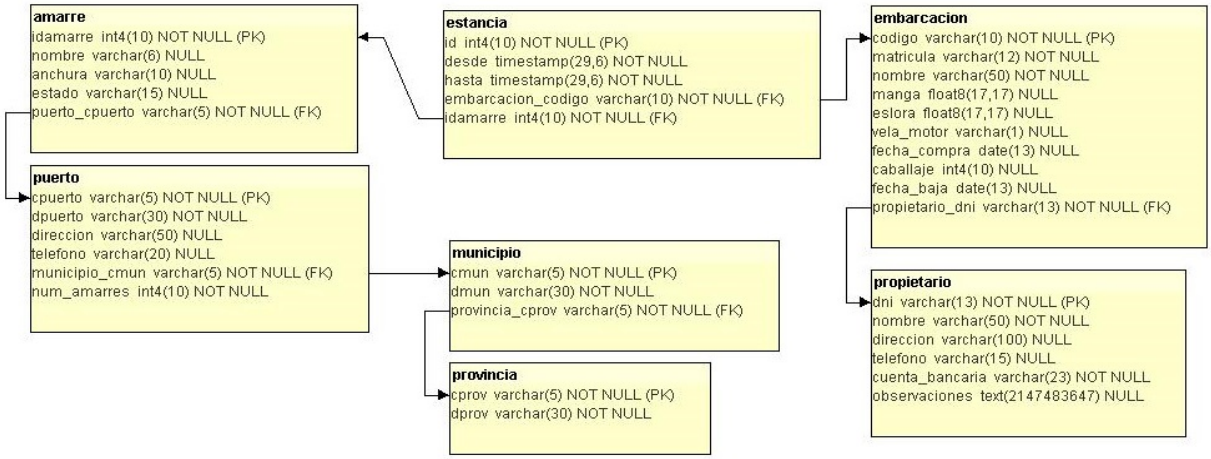
Se crea la entrada Jira de nuevo proyecto, y se deja en fase de “diagnóstico” para tratar de alcanzar un compromiso entre DGTI y empresa sobre las horas necesarias para acometer el proyecto.

En la empresa, para realizar la estimación, se realiza un borrador del documento técnico de análisis, que consiste un diagrama de clases (8h) y una breve redacción (3h).

<<La información que tenemos de cada embarcación es un código, una matrícula, un nombre, un propietario, las dimensiones (manga y eslora), si tiene vela o motor, el caballaje, y la fecha del último cambio de titular. Los atributos código, matrícula y nombre son únicos. De cada estancia sabemos, además del puerto y la embarcación, la fecha y hora de entrada y de salida y donde está amarrado, con la restricción de que un barco sólo puede estar en un momento dado en un puerto.>>



Se realiza también una estimación sobre las horas necesarias en la fase de diseño, para ello, se traduce el modelo de clases a un modelo relacional (1h) y se genera el DDL correspondiente (1h).




```
/* Creamos el Schema */
CREATE SCHEMA embarcaciones
  AUTHORIZATION postgres;
COMMENT ON SCHEMA demo
  IS 'Embarcaciones';

/* Creación del usuario propietario */
CREATE USER embarcaciones PASSWORD 'barco';
ALTER USER demo VALID UNTIL 'infinity';

CREATE USER embarcaciones_ae PASSWORD 'barco_ae';
ALTER USER embarcaciones_ae VALID UNTIL 'infinity';

CREATE USER embarcaciones_ac PASSWORD 'barco_ac';
ALTER USER embarcaciones_ac VALID UNTIL 'infinity';

/* Creación de los roles */
CREATE ROLE rembarcaciones ADMIN embarcaciones;
CREATE ROLE rembarcaciones_c ADMIN embarcaciones;

/* Asignar los roles a los usuarios */
GRANT rembarcaciones   TO embarcaciones;
GRANT rembarcaciones   TO embarcaciones_ae;
GRANT rembarcaciones_c TO embarcaciones_ac;

ALTER USER embarcaciones SET search_path TO embarcaciones,demo;
ALTER USER embarcaciones_ac SET search_path TO embarcaciones,demo;
ALTER USER embarcaciones_ae SET search_path TO embarcaciones,demo;


CREATE TABLE embarcaciones.propietario (
    dni character varying(13) NOT NULL,
    nombre character varying(50) NOT NULL,
    direccion character varying(100),
    telefono character varying(15),
    cuenta_bancaria character varying(23) NOT NULL,
    observaciones text,
    CONSTRAINT propietario_pkey PRIMARY KEY (dni),
    CONSTRAINT nombreuni UNIQUE (nombre),
    CONSTRAINT cuenta_bancariauni UNIQUE (cuenta_bancaria)
);
ALTER TABLE embarcaciones.propietario OWNER TO embarcaciones;


CREATE TABLE embarcaciones.provincia (
    cprov character varying(5) NOT NULL,
    dprov character varying(30) NOT NULL,
    CONSTRAINT provincia_pkey PRIMARY KEY (cprov),
    CONSTRAINT emb_dprovuni UNIQUE (dprov)
);
ALTER TABLE embarcaciones.provincia OWNER TO embarcaciones;


CREATE TABLE embarcaciones.municipio (
    cmun character varying(5) NOT NULL,
    dmun character varying(30) NOT NULL,
    provincia_cprov character varying(5) NOT NULL,
    CONSTRAINT municipio_pkey PRIMARY KEY (cmun),
```

```
    CONSTRAINT muni_dmununi UNIQUE (dmun),
            CONSTRAINT   fk_municipio_provincia1   FOREIGN   KEY   (provincia_cprov)   REFERENCES
embarcaciones.provincia(cprov) ON UPDATE CASCADE ON DELETE CASCADE
);
ALTER TABLE embarcaciones.municipio OWNER TO embarcaciones;


CREATE TABLE embarcaciones.embarcacion (
    codigo character varying(10) NOT NULL,
    matricula character varying(12) NOT NULL,
    nombre character varying(50) NOT NULL,
    manga double precision,
    eslora double precision,
    vela_motor character varying(1),
    fecha_compra date,
    caballaje integer,
    fecha_baja date,
    propietario_dni character varying(13) NOT NULL ,
    CONSTRAINT embarcacion_pkey PRIMARY KEY (codigo),
    CONSTRAINT emb_matriculauni UNIQUE (matricula),
    CONSTRAINT emb_nombreuni UNIQUE (nombre),
           CONSTRAINT   fk_embarcacion_propietario   FOREIGN   KEY   (propietario_dni)   REFERENCES
embarcaciones.propietario(dni)
);
ALTER TABLE embarcaciones.embarcacion OWNER TO embarcaciones;


CREATE TABLE embarcaciones.puerto (
    cpuerto character varying(5) NOT NULL,
    dpuerto character varying(30) NOT NULL,
    direccion character varying(50),
    telefono character varying(20),
    municipio_cmun character varying(5) NOT NULL,
    num_amarres integer NOT NULL,

    CONSTRAINT puerto_pkey PRIMARY KEY (cpuerto),
    CONSTRAINT puerto_dpuertouni UNIQUE (dpuerto),
               CONSTRAINT   fk_puerto_municipio1   FOREIGN   KEY   (municipio_cmun)   REFERENCES
embarcaciones.municipio(cmun)
);
ALTER TABLE embarcaciones.puerto OWNER TO embarcaciones;


CREATE TABLE embarcaciones.amarre (
    idamarre integer,
    nombre character varying(6),
    anchura character varying(10),
    estado character varying(15),
    puerto_cpuerto character varying(5) NOT NULL,

    CONSTRAINT idamarre_pkey PRIMARY KEY (idamarre),
    CONSTRAINT amarre_nombre UNIQUE (nombre),
                CONSTRAINT   fk_amarre_puerto   FOREIGN   KEY   (puerto_cpuerto)   REFERENCES
embarcaciones.puerto(cpuerto)
);
ALTER TABLE embarcaciones.amarre OWNER TO embarcaciones;


CREATE TABLE embarcaciones.estancia (
    id integer NOT NULL,
    desde timestamp without time zone NOT NULL,
    hasta timestamp without time zone NOT NULL,
    embarcacion_codigo character varying(10) NOT NULL,
    idamarre integer NOT NULL,

    CONSTRAINT estancia_pk PRIMARY KEY (id),
    CONSTRAINT fk_estancia_embarcacion
       FOREIGN KEY (embarcacion_codigo) REFERENCES embarcaciones.embarcacion(codigo)
               ON UPDATE CASCADE ON DELETE CASCADE,
    CONSTRAINT fk_estancia_amarre FOREIGN KEY (idamarre) REFERENCES embarcaciones.amarre(idamarre)
);
ALTER TABLE embarcaciones.estancia OWNER TO embarcaciones;


CREATE SEQUENCE embarcaciones.scmn_id_errlog
        START WITH 1
        INCREMENT BY 1
        NO MINVALUE
        NO MAXVALUE
```

```
        CACHE 1;
ALTER TABLE embarcaciones.scmn_id_errlog OWNER TO embarcaciones;
GRANT ALL ON SEQUENCE embarcaciones.scmn_id_errlog TO rembarcaciones;
SELECT pg_catalog.setval('embarcaciones.scmn_id_errlog', 1, true);

CREATE TABLE embarcaciones.tcmn_errlog (
    iderror numeric(9,0) NOT NULL,
    aplicacion character varying(30) NOT NULL,
    modulo character varying(30),
    version character varying(15) NOT NULL,
    usuario character varying(30) NOT NULL,
    fecha timestamp without time zone NOT NULL,
    tipo numeric(2,0) NOT NULL,
    mensaje text NOT NULL,
    CONSTRAINT tcmn_errlog_pkey PRIMARY KEY (iderror)
);
ALTER TABLE embarcaciones.tcmn_errlog OWNER TO rembarcaciones;

GRANT ALL ON SCHEMA embarcaciones TO embarcaciones;
GRANT USAGE ON SCHEMA embarcaciones TO rembarcaciones;
GRANT USAGE ON SCHEMA embarcaciones TO rembarcaciones_c;

REVOKE ALL ON TABLE embarcaciones.embarcacion FROM PUBLIC;
REVOKE ALL ON TABLE embarcaciones.embarcacion FROM embarcaciones;
GRANT ALL ON TABLE embarcaciones.embarcacion TO embarcaciones;
GRANT SELECT,INSERT,DELETE,UPDATE ON TABLE embarcaciones.embarcacion TO rembarcaciones;
GRANT SELECT ON TABLE embarcaciones.embarcacion TO rembarcaciones_c;

REVOKE ALL ON TABLE embarcaciones.estancia FROM PUBLIC;
REVOKE ALL ON TABLE embarcaciones.estancia FROM embarcaciones;
GRANT ALL ON TABLE embarcaciones.estancia TO embarcaciones;
GRANT SELECT,INSERT,DELETE,UPDATE ON TABLE embarcaciones.estancia TO rembarcaciones;
GRANT SELECT ON TABLE embarcaciones.estancia TO rembarcaciones_c;

REVOKE ALL ON TABLE embarcaciones.municipio FROM PUBLIC;
REVOKE ALL ON TABLE embarcaciones.municipio FROM embarcaciones;
GRANT ALL ON TABLE embarcaciones.municipio TO embarcaciones;
GRANT SELECT,INSERT,DELETE,UPDATE ON TABLE embarcaciones.municipio TO rembarcaciones;
GRANT SELECT ON TABLE embarcaciones.municipio TO rembarcaciones_c;

REVOKE ALL ON TABLE embarcaciones.propietario FROM PUBLIC;
REVOKE ALL ON TABLE embarcaciones.propietario FROM embarcaciones;
GRANT ALL ON TABLE embarcaciones.propietario TO embarcaciones;
GRANT SELECT,INSERT,DELETE,UPDATE ON TABLE embarcaciones.propietario TO rembarcaciones;
GRANT SELECT ON TABLE embarcaciones.propietario TO rembarcaciones_c;
REVOKE ALL ON TABLE embarcaciones.provincia FROM PUBLIC;
REVOKE ALL ON TABLE embarcaciones.provincia FROM embarcaciones;
GRANT ALL ON TABLE embarcaciones.provincia TO embarcaciones;
GRANT SELECT,INSERT,DELETE,UPDATE ON TABLE embarcaciones.provincia TO rembarcaciones;
GRANT SELECT ON TABLE embarcaciones.provincia TO rembarcaciones_c;

REVOKE ALL ON TABLE embarcaciones.puerto FROM PUBLIC;
REVOKE ALL ON TABLE embarcaciones.puerto FROM embarcaciones;
GRANT ALL ON TABLE embarcaciones.puerto TO embarcaciones;
GRANT SELECT,INSERT,DELETE,UPDATE ON TABLE embarcaciones.puerto TO rembarcaciones;
GRANT SELECT ON TABLE embarcaciones.puerto TO rembarcaciones_c;

REVOKE ALL ON SEQUENCE embarcaciones.scmn_id_errlog FROM PUBLIC;
REVOKE ALL ON SEQUENCE embarcaciones.scmn_id_errlog FROM rembarcaciones;

GRANT SELECT,USAGE,UPDATE ON TABLE embarcaciones.scmn_id_errlog TO rembarcaciones;
GRANT ALL ON SEQUENCE embarcaciones.scmn_id_errlog TO rembarcaciones;

REVOKE ALL ON TABLE embarcaciones.tcmn_errlog FROM PUBLIC;
REVOKE ALL ON TABLE embarcaciones.tcmn_errlog FROM rembarcaciones;
GRANT SELECT,INSERT,DELETE,UPDATE ON TABLE embarcaciones.tcmn_errlog TO rembarcaciones;
GRANT ALL ON TABLE embarcaciones.tcmn_errlog TO rembarcaciones;
```

Además del esquema, se prepara una carga de datos mínima

```
INSERT INTO embarcaciones.propietario
VALUES
(
   '1111111H',
   'Ignacio Santos Gaudioso',
```

```
   'C/Visitación 2 - Benetusser',
   '(+34)-666000111',
   '1111 1111 11 1111111111',
   'bla bla bla bla'
)
;
INSERT INTO embarcaciones.propietario
VALUES
(
   '0000001A',
   'Antonio Félix',
   'C/Lérida 21 - Foios',
   '(+34)-666000222',
   '2222 2222 22 2222222222',
   'bla bla bla bla'
)
;
INSERT INTO embarcaciones.propietario
VALUES
(
   '73775749Y',
   'Verónica Navarro',
   'C/Bilbao 10 - Valencia',
   '(+34)-666000333',
   '3333 2222 11 1232425222',
   'bla bla bla bla'
)
;
INSERT INTO embarcaciones.propietario
VALUES
(
   '73775735S',
   'David Pascual',
   'C/Sagunto 1 - Valencia',
   '(+34)-666000444',
   '2552 5252 65 2652232133',
   'bla bla bla bla'
)
;
INSERT INTO embarcaciones.embarcacion
VALUES
(
   '1', 'VA-3-128-05', 'Lébole', 1.6, 3.2, 'm', '2009-12-12', 10, NULL, '1111111H'
)
;
INSERT INTO embarcaciones.embarcacion
VALUES
(
   '2', 'AL-2-232-06', 'Mar', 2.2, 2.6, 'm', '2011-01-20', 10, NULL,'0000001A'
)
;
INSERT INTO embarcaciones.embarcacion
VALUES
(
   '3', 'VA-3-100-10', 'Aitana', 2.3, 3.3, 'v', '2010-10-21', 9, NULL,'73775749Y'
)
;
INSERT INTO embarcaciones.embarcacion
VALUES
(
   '4', 'VA-1-095-01', 'Luca', 2.6, 3, 'v', '2010-06-11', 9, NULL,'73775735S'
)
;
INSERT INTO embarcaciones.provincia
VALUES ('03', 'ALICANTE')
;
INSERT INTO embarcaciones.provincia
VALUES ('46', 'VALENCIA')
;
INSERT INTO embarcaciones.municipio
VALUES ('03739', 'Jávea', '03')
;
INSERT INTO embarcaciones.municipio
VALUES ('03700', 'Dénia', '03')
;
INSERT INTO embarcaciones.municipio
VALUES ('46520', 'Puerto de Sagunto', '46')
;
INSERT INTO embarcaciones.puerto
```

```
VALUES ('1', 'El portet de Dénia', 'Avd. Marítimo', '96 32 52 654', '03700',1)
;
INSERT INTO embarcaciones.puerto
VALUES ('2', 'Puerto Jávea', 'Avd. Mediterráneo', '96 36 52 636', '03739',3)
;
INSERT INTO embarcaciones.puerto
VALUES ('3', 'El port de Sagunt', 'Avd. Al Mar', '96 35 95 444', '46520',5)
;
INSERT INTO embarcaciones.amarre
VALUES (1, 'D-01A', 'GRANDE', 'CORRECTO', '1')
;
INSERT INTO embarcaciones.amarre
VALUES (2, 'D-02A', 'GRANDE', 'MANTENIMIENTO', '1')
;
INSERT INTO embarcaciones.amarre
VALUES (3, 'D-01B', 'MEDIO', 'CORRECTO', '1')
;
INSERT INTO embarcaciones.amarre
VALUES (4, 'J-01A', 'GRANDE', 'CORRECTO', '2')
;
INSERT INTO embarcaciones.amarre
VALUES (5, 'J-01B', 'MEDIO', 'CORRECTO', '2')
;
INSERT INTO embarcaciones.amarre
VALUES (6, 'J-02B', 'MEDIO', 'CORRECTO', '2')
;
INSERT INTO embarcaciones.amarre
VALUES (7, 'J-01C', 'PEQUEÑO', 'DESHABILITADO', '2')
;
INSERT INTO embarcaciones.amarre
VALUES (8, 'PS-01A', 'GRANDE', 'CORRECTO', '3')
;
INSERT INTO embarcaciones.amarre
VALUES (9, 'PS-02A', 'GRANDE', 'CORRECTO', '3')
;
INSERT INTO embarcaciones.amarre
VALUES (10, 'PS-01B', 'MEDIO', 'MANTENIMIENTO', '3')
;
INSERT INTO embarcaciones.estancia
VALUES (1,'2011-01-13 00:00:00', '2011-01-20 00:00:00', '2', 1)
;
INSERT INTO embarcaciones.estancia
VALUES (2,'2011-02-13 00:00:00', '2011-02-14 00:00:00', '2', 3)
;
INSERT INTO embarcaciones.estancia
VALUES (3,'2011-03-13 00:00:00', '2011-03-20 00:00:00', '3', 6)
;
INSERT INTO embarcaciones.estancia
VALUES (4,'2011-04-13 00:00:00', '2011-04-14 00:00:00', '4', 9)
;
```

Además, se realiza un estudio sobre el esquema de aplicación (distribución de pantallas) (6h) y se redacta el documento (1h).

La aplicación se va a descomponer en **6 ventanas funcionales**.

Mantenimiento de Propietarios
Patrón de interfaz: Registro (FIL-EDI)
Se debe tener en cuenta que los campos dni, cuenta y nombre son obligatorios. Se deben añadir máscaras en el registro (EDI) a teléfono y cuenta bancaria. El campo observaciones es un área de texto.

Mantenimiento de Provincias y Municipios
Patrón de interfaz: Maestro detalle Tabular-Tabular (FIL-LIS/LIS)
Las descripciones deben almacenarse en la BD siempre en mayúsculas

Mantenimiento de Embarcaciones
Patrón de interfaz: Tabular-Registro (FIL-LIS-EDI)
En el modo tabular no debe aparecer toda la información de la tabla. No interesa que aparezcan los siguientes campos manga, eslora, vela-motor y fecha de la compra La fecha de compra debe tener un calendario asociado El estado de la embarcación podrá ser activa o baja (lista estática).Cuando se cambie el estado de una embarcación a “baja” el sistema cumplimentará de forma automática el campo “fecha de baja”. Los campos código y matrícula solamente son editables al insertar y se mantendrán durante la vida de la embarcación. Habrá que validar en el alta que no existe otra embarcación con la misma matrícula. Los campos manga y eslora se introducen en metros con 2 decimales El propietario de la embarcación debe estar dado de alta en el sistema antes de dar de alta la embarcación, la forma de vincular una embarcación a un propietario será través de una ventana de selección. Una embarcación será siempre a remo, vela o motor, si es este ultimo caso, deberemos cumplimentar obligatoriamente el campo caballaje (en CV)

Mantenimiento de Puertos y Estancias
Patrón de interfaz: Maestro detalle (FIL-EDI/LIS)
Las provincias y municipios serán listas desplegables dependientes Los datos de la embarcación (nombre y matrícula) se completarán a través de una ventana de selección Deberá validarse que el barco no está registrado en otro puerto para esas fechas Los lunes no puede entrar ninguna embarcación nueva, el sistema deberá comprobarlo y advertir al usuario si procede

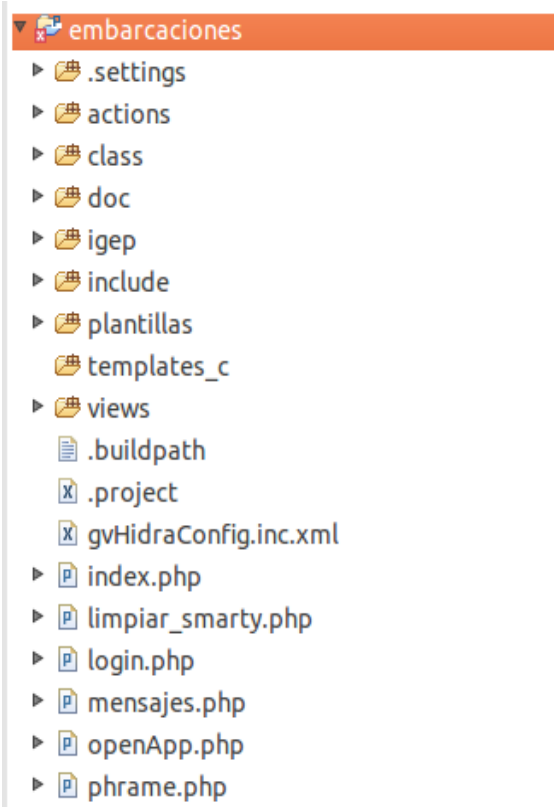
Mantenimiento de Embarcaciones y Estancias
Patrón de interfaz: Maestro detalle (FIL-LIS/LIS)
La búsqueda debe comprender rangos de fecha y/o embarcaciones El maestro sólo muestra datos, no puede realizar modificación, baja o alta alguna desde dicha pantalla El campo puertos es una lista con los puertos que tienen amarres libres. El campo amarres es una lista, dependiente de la de puertos, con la lista de amarres que están libres y en estado correcto para el puerto elegido.

Mantenimiento de Puertos y Amarres
Patrón de interfaz: Maestro / Detalle (FIL-LIS/LIS)
Los datos del puerto no pueden modificarse desde esta pantalla, simplemente se muestran Los datos de los amarres de un puerto si pueden modificarse. El estado de un amarre puede ser Correcto (c) En mantenimiento (m) y deshabilitado (d) para poder pasar a estados distintos del correcto, el amarre debe estar libre La anchura puede tener tres valores, grande (A), mediano (B) y pequeño (C) El nombre de un amarre incluye al final la letra correspondiente a la anchura

4.5.2 Solución

Para empezar necesitaremos bajarnos el proyecto gvHidra (***igep***[6]) y el proyecto genaro. Las descargas de ambos proyectos, incluyen el número de versión, Ej. ***igep-4_0_7***, por ello, hay que renombrar ambos proyectos y dejarlos sin versión, sólo con su nombre, es decir, gvHidra/igep y genaro.

Una vez nos hemos bajado ambos proyectos, en `igep/docs/appTemplate` se encuentra toda la estructura inicial que se requiere para empezar un proyecto gvHIDRA. Copiamos ese directorio, `appTemplate` fuera de la carpeta igep, y lo renombramos, en este caso por "embarcaciones". En el raíz de la carpeta "embarcaciones" copiamos el directorio igep, quedando como vemos en la imagen:



Por otra parte, dentro del directorio `include`, se copia el proyecto genaro. Llegados a este punto, debemos crear una carpeta `templates_c`, directorio donde la aplicación ubica la caché de las plantillas de la vista. Es importante que dicho directorio cuente con permisos de escritura para el usuario que ejecuta el servidor web (apache).

Tras seguir los pasos anteriores, ya podemos acceder a la primera pantalla de la aplicación, veremos una pantalla de validación con usuario y *password*, y un título de aplicación por defecto.

[6] Como ya hemos comentado ***igep*** son las siglas de "**I**mplementación ***G***uía de ***E***stilo ***P***HP", primer nombre de gvHidra, y carpeta que identifica dentro de una aplicación gvHidra el directorio donde se ubica el *"core"*.

Ahora ya podemos empezar a configurar la aplicación. Lo primero que debemos hacer es abrir el fichero, gvHidraConfig.inc.xml, se encuentra en el raíz de la aplicación. Al abrir el fichero veremos una explicación de la DTD del xml, si bajamos hasta encontrar la zona de configuración, veremos todo lo que tenemos que configurar correctamente.

Debemos configurar varios aspectos generales de la aplicación:

- Nombre de la aplicación `<applicationName>`
- Versión de la aplicación `<appVersion>`
- Título de la aplicación `<customTitle>`
- Directorio temporal para la aplicación `<templatesCompilationdir>`
- Hojas de estilo a utilizar en la aplicación `<customDirName>`
- Recarga de los mappings, fichero de redirección `<reloadMappings>`
- Compilación de las plantillas, no utilizar caché `<smartyCompileCheck>`
- Estado del fichero de log `<logSettings>`
- Elección del formato del where de las consultas `<queryMode>`
- Configuración de acceso a la/s BD/s `<DSNZone>...</DSNZone>`

```
<applicationName>embarcaciones</applicationName>
<appVersion>1.0.0</appVersion>
<customTitle>Embarcaciones</customTitle>
<templatesCompilationDir>templates_c/</templatesCompilationDir>
<customDirName>blueStyle</customDirName>
<reloadMappings>true</reloadMappings>
<smartyCompileCheck>true</smartyCompileCheck>
<logSettings status='LOG_ALL' dsnRef ='gvh_dsn_log'/>
<queryMode status='2' />
<DSNZone>
      <dbDSN id='g_dsn' sgbd='postgres'>
          <dbHost>gvhidra-test.gva.es</dbHost>
          <dbDatabase>embarcacionesdb</dbDatabase>
          <dbUser>gvh</dbUser>
          <dbPassword>gvh</dbPassword>
      </dbDSN>
      <dbDSN id='gvh_dsn_log' sgbd='postgres'>
           <dbHost>gvhidra-test.gva.es</dbHost>
```

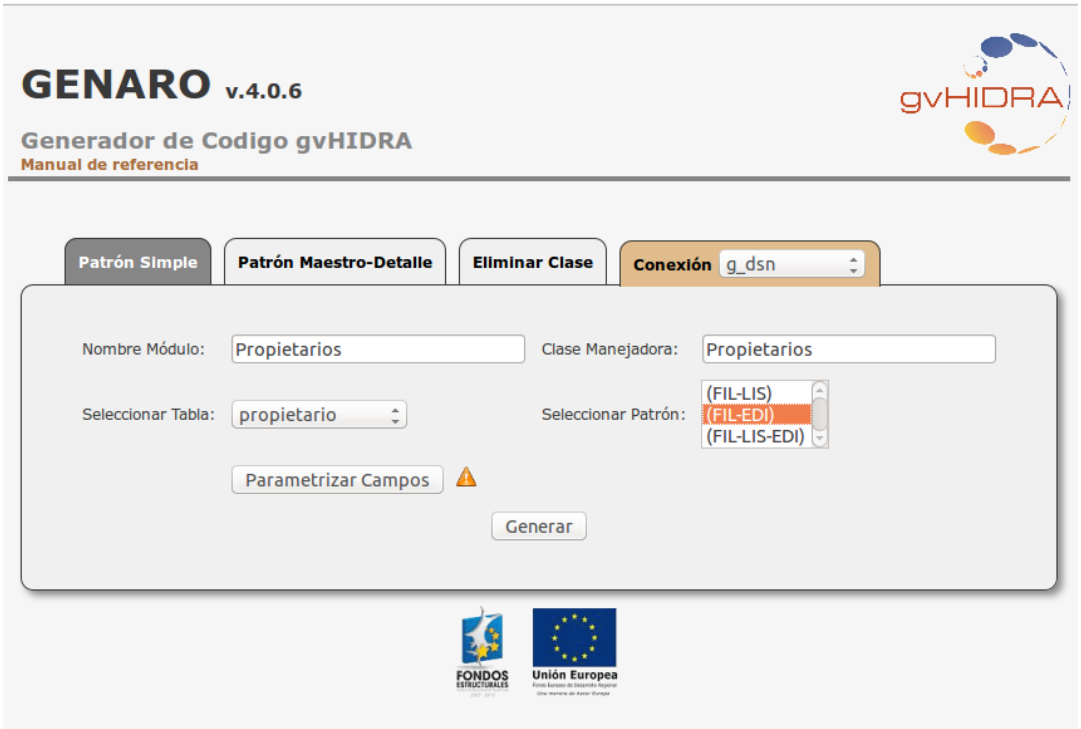
```
        <dbDatabase>embarcacionesdb</dbDatabase>
        <dbUser>gvh</dbUser>
        <dbPassword>gvh</dbPassword>
    </dbDSN>
```

Ya tenemos configurada la conexión de la BD y los parámetros generales de la aplicación. Ahora, si nos validamos ya accedemos a la pantalla principal de la aplicación.
En el bloque de “Menú módulos” no vemos ninguna opción, y en “Menú Herramientas”, ya vemos la opción “Genaro”.

Vamos a crear la primera pantalla con la herramienta Genaro

Mantenimiento de Propietarios
Patrón de interfaz: Registro (FIL-EDI)
Se debe tener en cuenta que los campos dni, cuenta y nombre son obligatorios. Se deben añadir máscaras en el registro (EDI) a teléfono y cuenta bancaria. El campo observaciones es un área de texto.

Una vez leído el enunciado, lanzamos Genaro y nos aparecerá la siguiente ventana:



Lo primero que hay que hacer es seleccionar la conexión a utilizar para crear las pantallas. El identificador de la conexión corresponde con el definido en el fichero `gvHidraConfig.inc.xml`, explicado antes, en nuestro caso será `g_dsn`.

Esta primera pantalla corresponde a un patrón simple según el enunciado, por lo tanto elegimos la solapa “Patrón Simple” y rellenamos los campos:

- Nombre módulo: se refiere al nombre del módulo que creará un nivel en el bloque de menú principal (bloque de menú que aparece a la izquierda de la pantalla principal). En nuestro caso lo llamamos “Propietarios”.

- Clase manejadora: Nombre que se le da a la clase que manejará la pantalla. En nuestro ejemplo la hemos llamado “Propietarios”.

- Tabla: Seleccionamos la tabla sobre la que trabajará la pantalla. En el ejemplo se va a trabajar sobre la tabla “propietario”.

- Patrón: Patrón que elegimos para realizar la pantalla. En este caso, el enunciado ya nos indica el tipo de panel, FIL-EDI.

- Parametrizar los campos: Pulsando el botón pasaremos a una pantalla donde podremos precisar más información, exactamente sobre los diferentes campos que aparecerán en el mantenimiento, como vemos en la imagen siguiente.

Aceptamos la configuración de los parámetros y le damos al botón “Generar”. Ya tenemos la primera pantalla creada, “Propietarios”. Si volvemos a entrar a la aplicación ya veremos la opción “Propietarios” en el menú Módulos.

Ya tenemos la pantalla generada, podría decirse que es un borrador del mantenimiento final. Campos distribuidos uno debajo de otro, sin organización, se muestran todos con tamaños de BD, y realizando las operaciones básicas. Ahora hay que distribuirlos y realizar las operaciones y comprobaciones extras que correspondan.

- Campos obligatorios: Esta restricción la hemos incluido en el diseño de la BBDD, por lo tanto no hay que tocar nada del código. De todos modos, vamos a comprobar como el generador ha marcado que estos campos son obligatorios al *framework* y este hace las validaciones pertinentes. Para ello abrimos la clase manejadora del mantenimiento `actions/Propietarios.php`, En el constructor de la clase “Propietarios” se definen los tipos de los campos, para comprobar la obligatoriedad del DNI, bajaremos hasta la zona de definición de tipos y veremos lo siguiente:

```
$string = new gvHidraString(false, 13); // No obligatorio
```

```
$this->addFieldType('fil_dni',$string);
$string = new gvHidraString(true, 13); // Obligatorio
$this->addFieldType('edi_dni',$string);
```

- **Máscaras:** ***<<Aplicar una máscara sobre los valores en los campos de texto teléfono y cuenta>>***

Para aplicar una máscara a un campo debemos hacerlo a partir de su tipo en la Clase Manejadora.
Para ello seguimos en la clase "Propietarios", en este caso buscamos la definición de tipo de datos de los campos teléfono (`fil_telefono`, `edi_telefono`), estas definiciones se encuentran en el constructor de la clase.
Tanto teléfono como la cuenta bancaria son definidos como tipo String (new `gvHidraString()`), antes de la asignación del tipo (`addFieldType`), aplicamos el método `setInputMask` a la variable `$string` asignándoles la máscara (+##)-######### (cada símbolo # representa un dígito en esa posición). Esta máscara tiene que afectar tanto al campo definido en el panel de búsqueda como en el panel registro (`fil_telefono`, `edi_telefono`).
Lo mismo hacemos para el tipo de los campos de la cuenta cuenta bancaria (`fil_cuenta_bancaria` y `edi_cuenta_bancaria`) con la siguiente máscara '#### #### ## ##########':

El código resultante es:

```
$string = new gvHidraString(false, 15);
$string->setInputMask('(+##)-#########');
$this->addFieldType('fil_telefono',$string);
$this->addFieldType('edi_telefono',$string);

$string = new gvHidraString(false, 23);
$string->setInputMask('#### #### ## ##########');
$this->addFieldType('fil_cuenta_bancaria',$string);
$this->addFieldType('edi_cuenta_bancaria',$string);
```

- **Áreas de texto:** ***<<el campo observaciones sea un área de texto en vez de un campo de texto simple>>***

Este campo observaciones no es muy útil en un panel de búsqueda, por lo tanto lo quitaremos. Para ello, tenemos que buscar en la plantilla de propietarios (plantillas/p_propietario.tpl) la zona donde se ubicará el panel de búsqueda, concretamente está comprendida entre las dos siguientes etiquetas:

```
{CWPanel id="fil" action="buscar" method="post" estado="$estado_fil"
claseManejadora="Propietarios"}
…
{/CWPanel}
```

Dentro de ese bloque encontraremos el siguiente código, que corresponde al campo Observaciones, y es el que queremos eliminar. Simplemente hay que borrarlo.

```
{CWCampoTexto textoAsociado="Observaciones" nombre="fil_observaciones" size="0"
maxlength="0" editable="true" value=$defaultData_Propietario.fil_observaciones
```

```
dataType=$dataType_Propietario.fil_observaciones}
```

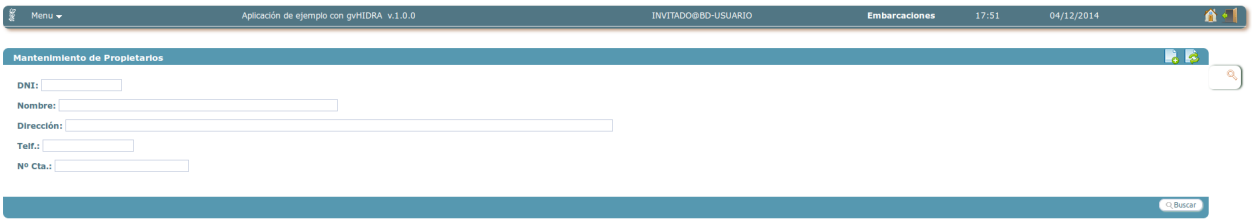
En el panel edición sí tiene sentido el campo, pero es más lógico que sea un campo tipo *Área de Texto*, que no un *Campo de Texto*. Si al generar la pantalla con Genaro no se ha tenido en cuenta esta elección de componente, lo tendremos como un *Campo de Texto*.
Abrimos la plantilla `p_propietario.tpl`, avanzamos hasta el bloque del panel edición "edi", que está comprendido entre las siguientes etiquetas:

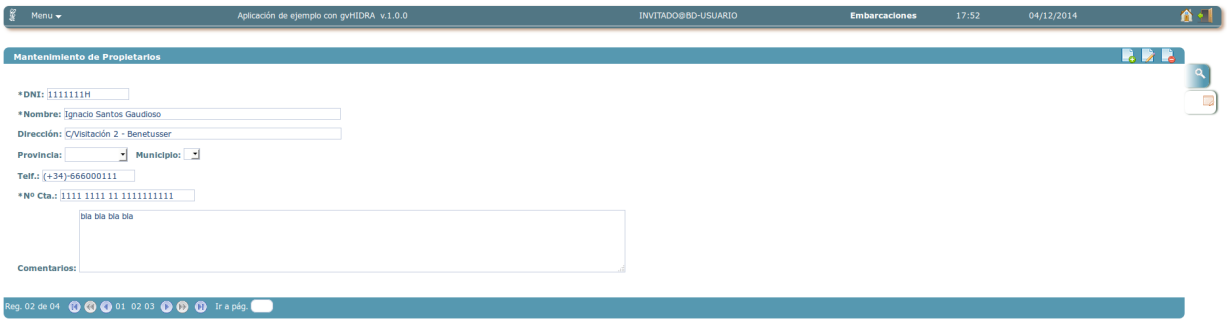
```
{CWPanel id="edi" tipoComprobacion="envio" action="operarBD" method="post"
estado="$estado_edi" claseManejadora="Propietarios"}
…
{/CWPanel}
```

Buscamos el CWCampoTexto correspondiente al campo Observaciones, y lo sustituimos por un área de texto, CWAreaTexto.

```
{CWCampoTexto textoAsociado="Observaciones" nombre="edi_observaciones" size="0"
maxlength="0" editable="true" value=$defaultData_Propietario.edi_observaciones
dataType=$dataType_Propietario.edi_observaciones}

sustituirlo por

{CWAreaTexto textoAsociado="Observaciones" nombre="edi_observaciones" rows="5"
cols="100" editable="true" value=$defaultData_Propietario.edi_observaciones
dataType=$dataType_Propietario.edi_observaciones}
```

Finalmente el mantenimiento nos queda como vemos en las imágenes:





MANTENIMIENTO DE PROVINCIAS Y MUNICIPIOS
Patrón de interfaz: Maestro detalle Tabular-Tabular (FIL-LIS/LIS)
Las descripciones deben almacenarse en la BD siempre en mayúsculas

Volvemos a lanzar Genaro para crear nuevamente esta pantalla. En este caso elegiremos la solapa “Patrón Maestro-Detalle”, donde rellenamos la información sobre el maestro (tabla, nombre de clase, número de detalles...) y la del detalle, a destacar que hay que seleccionar la clave ajena del detalle al maestro.

Los datos que hay que introducir en el maestro son como los que hemos introducido para los paneles simples en los ejercicios anteriores (nombre de clase manejadora, tabla sobre la que trabajará y tipo de panel), datos concretos de este panel son:

- Clave primaria del maestro, que será con la que se relacionará con el detalle.
- Número de detalles, con ello indicaremos cuantos detalles necesitamos, en nuestro ejemplo solamente es uno.

Bajo tenemos la zona para configurar el detalle, a destacar está el campo Clave Ajena, con el que se relacionará con el maestro.

Una vez rellenados todos los campos, damos a Generar y ya tendremos el maestro-detalle Provincias y Municipios creado. Tal y como hemos dicho en el ejercicio anterior, se crean los diferentes paneles sin una distribución de los campos más organizada, aparecen uno debajo de otro, el diseño del panel se puede hacer utilizando algunos etiquetas html para organizarlos, como tablas (<table>) o agrupadores de campos (<fieldset>) El siguiente código organiza los campos del panel búsqueda utilizando una tabla html:

```
{CWPanel id="fil" action="buscar" method="post" estado="$estado_fil" claseManejadora="Propietario"}
        {CWBarraSupPanel titulo="Mantenimiento de Propietario"}
                {CWBotonTooltip imagen="01" titulo="Insertar Propietario" funcion="insertar"
actuaSobre="ficha" action="Propietario__nuevo"}
                {CWBotonTooltip imagen="04" titulo="Limpiar campos" funcion="limpiar" actuaSobre="ficha"}
        {/CWBarraSupPanel}
        {CWContenedor}
                {CWFicha}
                <table class="text" cellspacing="4" cellpadding="4" border="0">
                <tr><td>
                {CWCampoTexto textoAsociado="DNI" nombre="fil_dni" size="13" editable="true" visible="true"
value=$defaultData_Propietario.fil_dni dataType=$dataType_Propietario.fil_dni}
                </td></tr>
                <tr><td>
                {CWCampoTexto textoAsociado="Nombre" nombre="fil_nombre" size="30" editable="true"
visible="true" value=$defaultData_Propietario.fil_nombre dataType=$dataType_Propietario.fil_nombre}
                </td></tr>
                </table>
                <br/>
                {/CWFicha}
        {/CWContenedor}
        {CWBarraInfPanel}
        {CWBoton imagen="50" texto="Buscar" class="button" accion="buscar"}
        {/CWBarraInfPanel}
{/CWPanel}
```

- **La descripción se almacena en la BD en mayúsculas.**

Por decisión de estilo, a la hora de almacenar los nombres de las provincias o municipios en mayúsculas o minúsculas, vamos a forzar que, independientemente, de lo que se haya escrito en el campo, en la BD se guarde siempre en mayúsculas. Este requisito requiere de una acción de interfaz, es decir, después de escribir un valor en el campo y abandonar el mismo, se ejecutará la acción que pasará el contenido del campo a mayúsculas. Para ello hay que seguir los siguientes pasos:

Vamos a la plantilla, `p_provincias.tpl`, buscamos el campo de descripción de la provincia (lisDpro). Una vez en él, hay que indicar que el contenido del campo se actualizará. Este evento se indica añadiendo el parámetro `actualizaA` en la definición del campo en la tpl.

```
{CWCampoTexto nombre="lis_dprov" editable="true" size="25" textoAsociado="Provincia"
dataType=$dataType_Provincias.lisDpro actualizaA="lis_dprov"}
```

Pasamos a la parte que llevará a cabo la acción, vamos a la clase Propietarios.php. En el constructor tenemos que indicar que el campo “lis_dprov” lanza la acción de interfaz, para ello está el método `addTriggerEvent(campoDisparador, nombreFunción)`. Esta acción se realiza en un método que debemos definir, por ejemplo, “*upperProvincia*”.

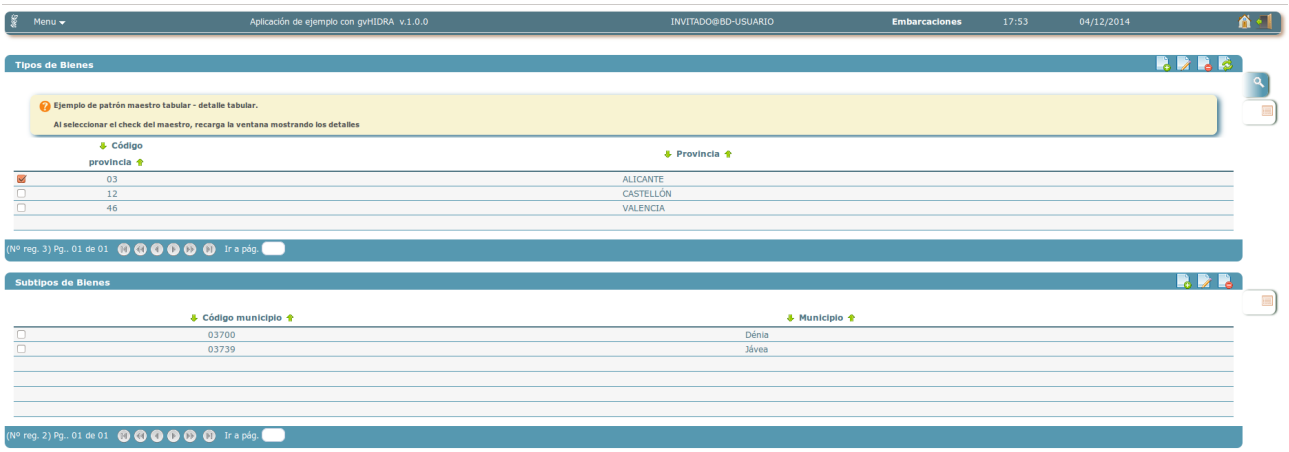
```
$this->addTriggerEvent("lis_dprov", "upperProvincia");
```

Implementamos la función “`upperProvincia()`” que realizará la acción que necesitamos.

```
public function upperProvincia($objDatos)
{
```

```
        $provincia = $objDatos->getValue('lis_dprov');
        $objDatos->setValue('lis_dprov',strtoupper($provincia));
        return 0;
}
```

Aquí podemos ver que las provincias ya aparecen en mayúsculas. Ahora debemos realizar lo mismo para los municipios.



Vamos a la plantilla `p_municipios.tpl` y ahora buscamos el campo "*lisDmun*", añadimos el parámetro que nos indica que hay una acción de interfaz para ese campo.

```
{CWCampoTexto nombre="lis_dmun" editable="true" size="25" textoAsociado="Municipio"
dataType=$dataType_Municipios.lisDmun actualizaA="lis_dmun"}
```

En la clase *Municipios.php* debemos hacer lo mismo que en provincias. Definir en el constructor qué función lanza el campo "**lisDmun**" e implementar la función.

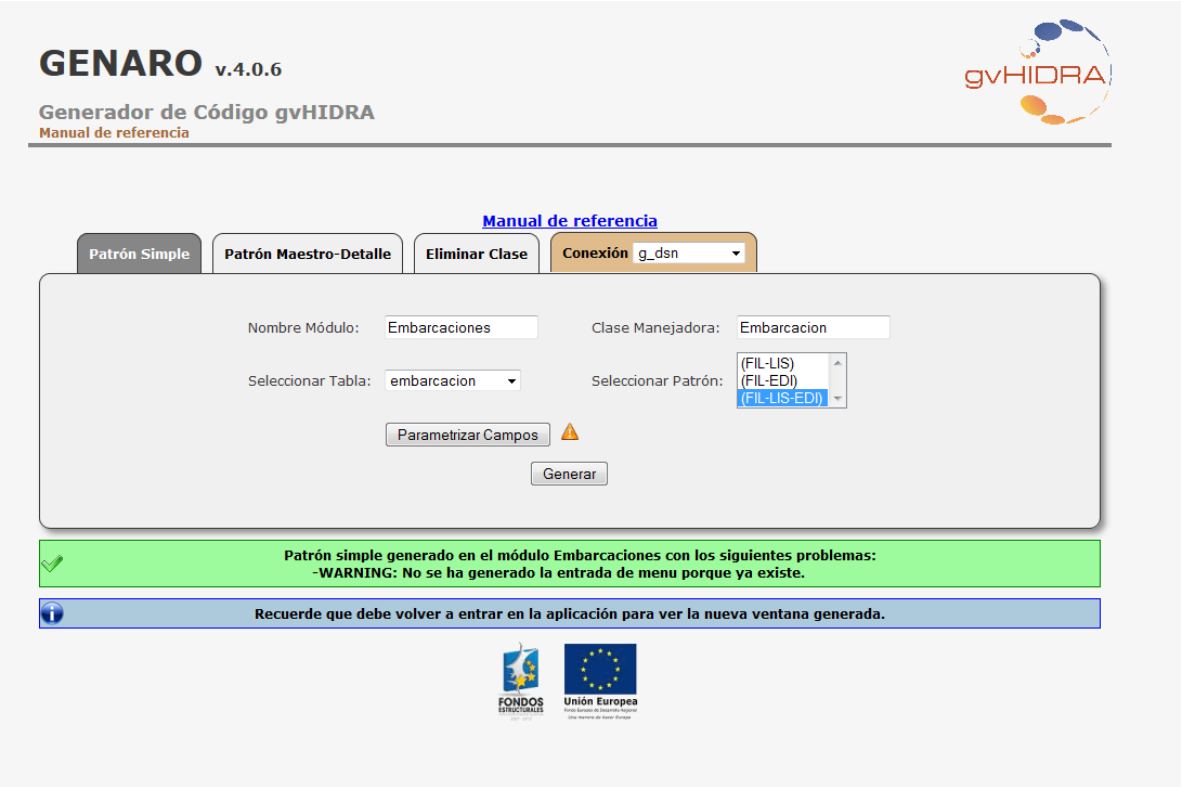
```
$this->addTriggerEvent("lis_dmun", "upperMunicipio");
```

Implementamos la función "`upperMunicipio()`" que realizará la acción que necesitamos.

```
public function upperMunicipio($objDatos)
{
        $municipio = $objDatos->getValue('lis_dmun');
        $objDatos->setValue('lis_dmun',strtoupper($municipio));
        return 0;
}
```

MANTENIMIENTO DE EMBARCACIONES
Patrón de interfaz: Tabular-Registro(FIL-LIS-EDI)
- En el modo tabular no debe aparecer toda la información de la tabla. No interesa que aparezcan los siguientes campos *manga, eslora, vela-motor y fecha de la compra.* - En el modo edición, la fecha de compra debe tener un calendario asociado. - El estado de la embarcación podrá ser activa o baja (lista estática).Cuando se cambie el estado de una embarcación a “baja” el sistema cumplimentará de forma automática el campo “fecha de baja”. - Los campos código y matrícula solamente se pueden editar al insertar y se mantendrán durante la vida de la embarcación. - Habrá que validar en el alta que no existe otra embarcación con la misma matrícula. - Los campos manga y eslora se introducen en metros con 2 decimales - El propietario de la embarcación debe estar dado de alta en el sistema antes de dar de alta la embarcación, la forma de vincular una embarcación a un propietario será través de una ventana de selección. - Una embarcación será siempre a remo, vela o motor, si es este ultimo caso, deberemos cumplimentar el campo caballaje de forma obligatoria (en CV)

Volvemos a lanzar Genaro para crear esta pantalla, en este caso el patrón simple que debemos elegir es el FIL-LIS-EDI, rellenar la clase manejadora, indicar la tabla, etc. Tal y como vemos en la imagen:



- **Simplificar la información a mostrar en el panel tabular.**

La información sobre eslora, manga, vela-motor y fecha de la compra, no queremos visualizarla en el panel tabular. Esta tarea es fácil, porque simplemente tenemos que quitar código.

Empezaremos por la plantilla, `plantillas/Embarcaciones/p_Embarcacion.tpl`. Buscamos la zona correspondiente al panel tabular:

```
{CWPanel id="lis" action="borrar" method="post" estado="$estado_lis"
claseManejadora="Embarcacion"}
...
{/CWPanel}
```

En ese bloque vemos todos los campos que Genaro ha creado basándose en los campos de la tabla "`embarcacion`", ahora solamente tenemos que eliminar aquellos que no queremos mostrar en el panel tabular. Exactamente son los siguientes:

```
{CWCampoTexto textoAsociado="Manga" nombre="lis_manga" size="8" editable="true" visible="true"
value=$defaultData_Embarcacion.lis_manga dataType=$dataType_Embarcacion.lis_manga}

{CWCampoTexto textoAsociado="Eslora" nombre="lis_eslora" size="8" editable="true" visible="true"
value=$defaultData_Embarcacion.lis_eslora dataType=$dataType_Embarcacion.lis_eslora}

{CWCampoTexto textoAsociado="Vela motor" nombre="lis_vela_motor" size="1" editable="true"
visible="true" value=$defaultData_Embarcacion.lis_vela_motor
dataType=$dataType_Embarcacion.lis_vela_motor}

{CWCampoTexto textoAsociado="Fecha compra" nombre="lis_fecha_compra" size="12" editable="true"
visible="true" value=$defaultData_Embarcacion.lis_fecha_compra
dataType=$dataType_Embarcacion.lis_fecha_compra}
```

El panel tabular nos debe quedar de la siguiente forma:

Para hacerlo bien, ya que esos campos no estarán ya en el panel tabular, también debemos eliminar su definición en el constructor de la clase manejadora `Embarcacion.php`.

Lo primero será eliminar en la zona de *matching*, lo correspondiente a los campos eliminados:

```
$this->addMatching("lis_manga","manga","embarcacion");
$this->addMatching("lis_eslora","eslora","embarcacion");
$this->addMatching("lis_vela_motor","vela_motor","embarcacion");
$this->addMatching("lis_fecha_compra","fecha_compra","embarcacion");
```

También eliminar la parte de definición de los tipos correspondientes a los campos del tabular:

```
$string = new gvHidraString(false, 1);
$this->addFieldType('fil_vela_motor',$string);
$this->addFieldType('lis_vela_motor',$string);  // ELIMINAR ESTA LÍNEA
$this->addFieldType('edi_vela_motor',$string);

$fecha = new gvHidraDate(false);
```

```
$fecha->setCalendar(false);
$this->addFieldType('fil_fecha_compra',$fecha);
$this->addFieldType('lis_fecha_compra',$fecha);  // ELIMINAR ESTA LÍNEA
$this->addFieldType('edi_fecha_compra',$fecha);

$float = new gvHidraFloat(false, 8);
$float->setFloatLength();
$this->addFieldType('fil_manga',$float);
$this->addFieldType('lis_manga',$float);  // ELIMINAR ESTA LÍNEA
$this->addFieldType('edi_manga',$float);

$float = new gvHidraFloat(false, 8);
$float->setFloatLength();
$this->addFieldType('fil_eslora',$float);
$this->addFieldType('lis_eslora',$float);  // ELIMINAR ESTA LÍNEA
$this->addFieldType('edi_eslora',$float);
```

- **En el panel edición, la fecha de compra debe llevar un calendario**

Para ello tenemos que ir a la clase manejadora `Embarcacion.php`, y en el constructor, donde se definen los tipos buscar el correspondiente a la fecha de compra. Por defecto vemos que se ha definido un mismo tipo Date para los campos del panel de búsqueda y edición, como queremos un calendario en el panel de edición crearemos un tipo nuevo para el campo edi_fecha_compra

```
$fecha = new gvHidraDate(false);
$fecha->setCalendar(false);
$this->addFieldType('fil_fecha_compra',$fecha);
$this->addFieldType('edi_fecha_compra',$fecha);


quedará así:


$fecha = new gvHidraDate(false);
$fecha->setCalendar(false);
$this->addFieldType('fil_fecha_compra',$fecha);
$fechaEdicion = new gvHidraDate(false);
$fechaEdicion->setCalendar(true);
$this->addFieldType('edi_fecha_compra',$fechaEdicion);
```

Simplemente es pasar un true al método `setCalendar()`. Ahora ya nos aparecerá un botón de calendario al lado del campo correspondiente a la fecha de compra.

- **Estado de embarcación, una lista estática con dos valores: activa o baja. Al ponerla en estado baja, la fecha de baja se autocompleta.**

Empezamos definiendo la lista estática de estado. Por ser una lista estática se define en el propio constructor de la clase manejadora, `Embarcacion.php`.

```
$estado = new gvHidraList('edi_estado');
$estado->addOption('','');
$estado->addOption('A','Activo');
$estado->addOption('B','Baja');
$estado->setSelected('');
$this->addList($estado);
```

Una cosa a tener en cuenta, es que en la consulta del panel edición hay que añadir “lis_estado” como un campo artificial, tal y como vemos a continuación:

```
$str_select_editar = "SELECT codigo as \"edi_codigo\", matricula as \"edi_matricula\",
nombre as \"edi_nombre\", manga as \"edi_manga\", eslora as \"edi_eslora\", vela_motor
as \"edi_vela_motor\", fecha_compra as \"edi_fecha_compra\", caballaje
as \"edi_caballaje\", fecha_baja as \"edi_fecha_baja\", propietario_dni
as \"edi_propietario_dni\",
 '' as \"edi_estado\"
FROM embarcacion";
```

Por otra parte, hay que definir el componente lista en la plantilla. Así que abrimos el fichero `p_Embarcacion.tpl` y añadimos la lista al lado del campo `edi_fecha_baja`:

```
{CWLista textoAsociado="Estado" nombre="edi_estado" size="30" editable="true"
visible="true" value=$defaultData_Embarcacion.edi_estado
dataType=$dataType_Embarcacion.edi_estado}
{CWCampoTexto textoAsociado="Fecha baja" nombre="edi_fecha_baja" size="12"
editable="true" visible="true" value=$defaultData_Embarcacion.edi_fecha_baja
dataType=$dataType_Embarcacion.edi_fecha_baja}
```

Hasta aquí ya tenemos la lista desplegable con los valores Activa y Baja. Ahora nos queda realizar la acción de interfaz, es decir, cuando en la lista se elija la opción “Baja” automáticamente se rellene el campo `edi_fecha_baja` con la fecha actual.

Lo primero que hay que hacer es indicar en la plantilla (`p_Embarcacion.tpl`) que la lista tiene asociada una acción de interfaz, como en el ejemplo de Provincias y Municipios, hay que añadir el parámetro “`actualizaA`”:

```
{CWLista textoAsociado="Estado" nombre="edi_estado" size="30" editable="true"
visible="true" value=$defaultData_Embarcacion.edi_estado
dataType=$dataType_Embarcacion.edi_estado actualizaA="edi_fecha_baja"}
```

Ahora hay que definir en la clase manejadora (`Embarcacion.php`) la acción de interfaz y su asociación al campo “`edi_estado`”. En el constructor se asocia el evento al campo:

```
$this->addTriggerEvent('edi_estado','addFechaBaja');
```

Después hay que definir el método “`addFechaBaja`”. El método realizará la comprobación del valor elegido en la lista (Activo – A o Baja – B), en el caso de ser Activo, se deshabilitará el campo `edi_fecha_baja` ya que no tiene sentido introducir esa fecha, en cambio, si se elige Baja, se completará de forma automática con la fecha del día.

```
public function addFechaBaja($objDatos)
{
	$estado = $objDatos->getValue('edi_estado');
	if ($estado == 'B')
	{
		$hoy = new DateTime ('today');
		$objDatos->setValue('edi_fecha_baja',$hoy);
	}
	else {
		$objDatos->setEnable('edi_fecha_baja',false);
```

```
        }
        return 0;
}
```

Quedando así la pantalla:

Los campos código y matrícula solamente se pueden editar al insertar y se mantendrán durante la vida de la embarcación.

Este punto es sencillo, solamente tenemos que acceder a la plantilla, `p_Embarcacion.tpl`. Una vez en ella ir a la zona de edición y buscar los campos “código” y “matrícula”. Lo único que tenemos que cambiar es el parámetro editable de ambos campos, por defecto viene a true, pero como solo queremos que se active cuando se vaya a insertar una nueva embarcación, lo tenemos que cambiar por “nuevo”

```
{CWCampoTexto textoAsociado="Codigo" nombre="edi_codigo" size="10" editable="nuevo"
visible="true" value=$defaultData_Embarcacion.edi_codigo
dataType=$dataType_Embarcacion.edi_codigo}

{CWCampoTexto textoAsociado="Matricula" nombre="edi_matricula" size="12"
editable="nuevo" visible="true" value=$defaultData_Embarcacion.edi_matricula
dataType=$dataType_Embarcacion.edi_matricula}
```

Ahora los campos aparecerán rellenados y deshabilitados cuando editamos la información de una embarcación existente.

- **Comprobar que la matrícula de una embarcación no existe ya**.

Lógicamente no puede haber dos embarcaciones con la misma matrícula, por lo tanto, cuando vayamos a insertar una embarcación nueva comprobaremos que no existe ya otra con esa matrícula. Hay dos formas de comprobar este requisito, una sería a través de una acción de interfaz que se compruebe si la matrícula introducida ya existe o no, y la otra, sería antes de realizar el INSERT en BD. Como la opción de la acción de interfaz ya se ha practicado en ejercicios anteriores, vamos a utilizar la segunda opción. Para ello vamos a la clase manejadora `Embarcacion.php`, buscamos el método `preInsert()`. Este método se ejecutará justo antes de hacer el INSERT de la embarcación, si ocurre que ya existe la matrícula podremos interrumpir el hilo de ejecución y no se insertará.

```
public function preInsertar($objDatos)
{
	$matricula = $objDatos->getValue('edi_matricula');
	$sql = <<<query
		SELECT
			*
		FROM
			embarcacion
		WHERE
			matricula = $matricula
query;
	$resultado = $this->consultar($sql);
	if (count($resultado) > 0)
	{
		$this->showMessage('APL-2');
		return -1;
	}
	return 0;
}
```

Vemos en el código que si la consulta nos devuelve datos mostramos un mensaje (`showMessage()`). Este tipo de mensajes se definen en el fichero `mensajes.php` que se encuentra en el raíz de la aplicación. La definición del nuestro es la siguiente:

```
'APL-2'=>array('descCorta'=>'Matrícula ya existente.','descLarga'=>'La matrícula
introducida ya pertenece a otra embarcación.','tipo'=>'ERROR'),
```

Los campos manga y eslora se introducen en metros con 2 decimales

Este requisito solamente requiere que se definan los decimales en la definición de tipos que se hace en el constructor, a través del método `setFloatLength()`.

```
$float = new gvHidraFloat(false, 8);
$float->setFloatLength(2);
$this->addFieldType('fil_manga',$float);
$this->addFieldType('edi_manga',$float);

$float = new gvHidraFloat(false, 8);
$float->setFloatLength(2);
this->addFieldType('fil_eslora',$float);
this->addFieldType('edi_eslora',$float);
$resultado = $this->consultar($sql);
```

- **El propietario de la embarcación debe estar dado de alta en el sistema antes de dar de alta la embarcación.**

Para asegurarnos de que un propietario ya está dado de alta en la aplicación es que la vinculación del mismo a la embarcación se haga a través de una ventana de selección. El campo dni del propietario se rellenará a través de la selección de un propietario de los que aparezcan en la ventana de selección. Lo primero para definir una ventana de selección es tener la consulta definida en el fichero `actions/principal/AppMainWindow.php`.

Definimos la consulta que necesitamos para obtener los propietarios y sus DNIs. Se le da un nombre (VAI_Propietarios) que es a través del cual se le hará referencia en la clase manejadora. Y por último, el array final contendrá los campos por los que se podrá buscar en la ventana de selección.

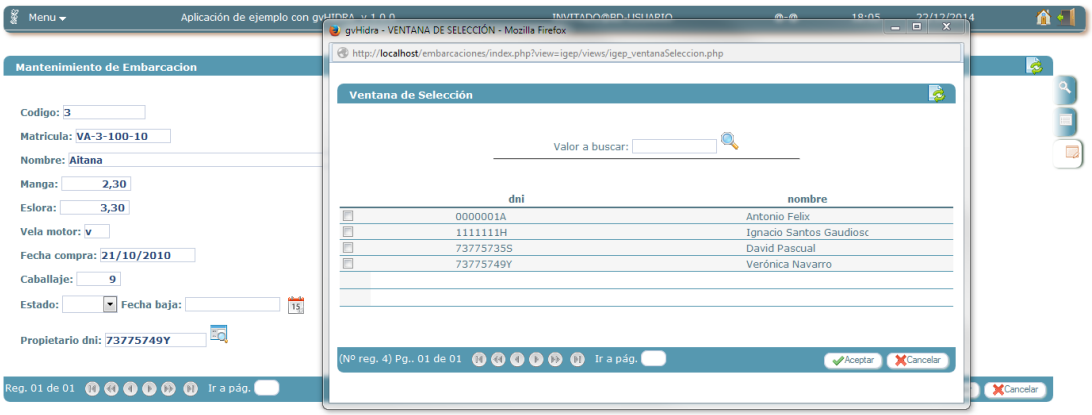
```
$sql = <<<query
       SELECT
              dni,
              nombre
       FROM
              propietario
query;
$conf->setSelectionWindow_DBSource('VAI_Propietarios', $sql, array("nombre"));
```

Una vez definida ya podemos hacer referencia a ella desde la clase manejadora (`Embarcacion.php`). Definimos la ventana de selección asociada al campo `edi_propietario_dni`, y asociamos el campo dni de la ventana de selección con el campo `edi_propietario_dni`. Podemos indicarle el número de filas por página que queremos visualizar y el tamaño de la ventana.

```
$propietario = new gvHidraSelectionWindow('edi_propietario_dni', 'VAI_Propietarios');
$propietario->addMatching('edi_propietario_dni','dni');
$propietario->setRowsNumber(6);
$propietario->setSize(800,600);
$this->addSelectionWindow($propietario);
```

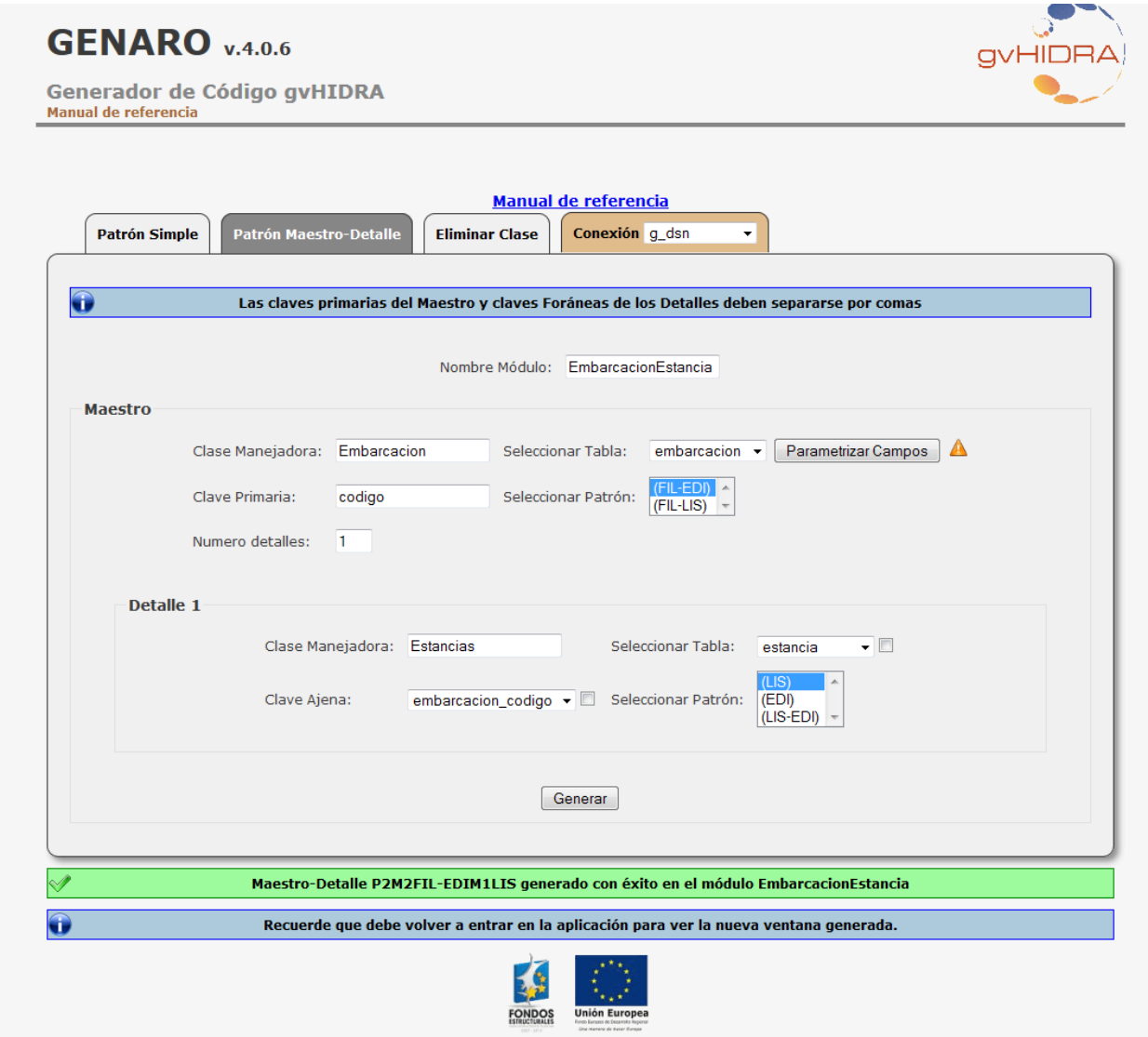
Y por último, en la plantilla `p_Embarcacion.tpl`, indicamos que debe aparecer un botón tooltip (`CWBotonTooltip`) que nos abrirá esta ventana de selección, el campo al que se asocia el botón viene dado por el parámetro `actuaSobre`.

```
{CWCampoTexto textoAsociado="Propietario dni" nombre="edi_propietario_dni" size="13"
editable="true" visible="true" value=$defaultData_Embarcacion.edi_propietario_dni
dataType=$dataType_Embarcacion.edi_propietario_dni}

{CWBotonTooltip imagen="13" titulo="Propietario" funcion="abrirVS"
actuaSobre="edi_propietario_dni"}
```



MANTENIMIENTO DE EMBARCACIONES Y ESTANCIAS
Patrón de interfaz: Maestro-Detalle (FIL-LIS/LIS)
- El maestro sólo muestra datos, no puede realizar modificación, baja o alta alguna desde dicha pantalla - En el detalle no es necesario que sean visibles las claves primarias de la tabla Estancia, es decir el campo Id y el campo embarcacion_codigo. - El campo puertos es una lista con los puertos que tienen amarres libres. El campo amarres es una lista, dependiente de la de puertos, con la lista de amarres que están libres y en estado correcto para el puerto elegido. - Los campos fecha (desde y hasta) de las estancias deben llevar un calendario.

Generamos la pantalla:



- **El maestro solo muestra datos, no es posible ninguna modificación, eliminación o inserción.**

Esto es sencillo, solamente tenemos que abrir la plantilla (`plantillas/EmbarcacionEstancia/p_Embarcacion.tpl`) buscar la zona del panel edición, que está comprendido entre las siguientes etiquetas:

```
{CWPanel id="edi" esMaestro="true" itemSeleccionado=$smty_filaSeleccionada
action="operarBD" method="post" estado=$estado_edi claseManejadora="Embarcacion"
accion=$smty_operacionFichaEmbarcacion}
…
{/CWPanel}
```

Una vez encontrado hay que cambiar el valor del parámetro editable de todos los campos, hay que ponerlo a false, tal y como se ve a continuación:

```
{CWCampoTexto textoAsociado="Codigo" nombre="edi_codigo" size="10" editable="false"
visible="true" value=$defaultData_Embarcacion.codigo
dataType=$dataType_Embarcacion.codigo}
```

Y por otra parte, ya que no se va a poder hacer ningún tipo de trabajo sobre embarcación quitaremos los botones `tooltip` que aparecen en la barra superior del panel. Esto simplemente se hace eliminando las siguientes líneas:

```
{CWBotonTooltip imagen="01" titulo="Insertar registro" funcion="insertar"
actuaSobre="ficha" action="Embarcacion__nuevo"}
{CWBotonTooltip imagen="02" titulo="Modificar registro" funcion="modificar"
actuaSobre="ficha"}
{CWBotonTooltip imagen="03" titulo="Eliminar registro" funcion="eliminar"
actuaSobre="ficha"}
{CWBotonTooltip imagen="04" titulo="Restaurar valores" funcion="limpiar"
actuaSobre="ficha"}
```

Ahora, el maestro nos quedará con el siguiente aspecto:

- **Los campos clave de la tabla Estancia no deben ser visibles (campo ID y embarcacion_codigo)**

Este punto es sencillo, simplemente tenemos que abrir la plantilla (`plantillas/EmbarcacionEstancia/p_Embarcacion.tpl`), la zona del detalle está comprendida entre las siguientes etiquetas:

```
{if count($smty_datosFichaM) gt 0}
{CWPanel id="lisDetalle" detalleDe="edi" tipoComprobacion="envio" action="operarBD"
method="post" estado="on" claseManejadora="Estancias"}

…

{CWContenedorPestanyas id="Detalle"}
```

```
            {CWPestanya tipo="lis" panelAsociado="lisDetalle" estado="on"}
        {/CWContenedorPestanyas}
        {/if}
```

Una vez ahí, buscamos los campos indicados y le añadimos el parámetro `oculto="true"`, eliminamos los parámetros size, editable y visibilidad, pues no son necesarios si el campo es oculto.

```
{CWCampoTexto nombre="lis_id" oculto="true" value=$defaultData_Estancias.lis_id
dataType=$dataType_Estancias.lis_id}
{CWCampoTexto nombre="lis_embarcacion_codigo" oculto="true"
value=$defaultData_Estancias.lis_embarcacion_codigo
dataType=$dataType_Estancias.lis_embarcacion_codigo}
```

- **En el detalle Estancias, mostrar el puerto al que pertenece el amarre y el nombre del amarre. Solo se mostrarán los puertos que tienen amarres libres.**

Para ello necesitamos modificar la consulta que nos ha generado por defecto Genaro. Necesitamos añadir información, como el código del puerto y la relación entre estancia, amarre y puerto. La consulta quedaría así:

```
$str_select = <<<query
SELECT
id as "lis_id",
desde as "lis_desde",
hasta as "lis_hasta",
embarcacion_codigo as "lis_embarcacion_codigo",
estancia.idamarre as "lis_idamarre",
cpuerto as "lis_puerto"
FROM
estancia, puerto, amarre

query;

$this->setSelectForSearchQuery($str_select);

$str_where = "puerto.cpuerto = amarre.puerto_cpuerto AND amarre.idamarre =
estancia.idamarre";

$this->setWhereForSearchQuery($str_where);
```

El campo puerto no existe en el detalle, así que lo vamos a añadir, será una lista con los puertos que tienen amarres libres. Además el campo amarre se convertirá también en otra lista, lista de los amarres disponibles del puerto seleccionado en la lista anterior. En definitiva serán dos listas dependientes.
En la plantilla (`plantillas/EmbarcacionEstancia/p_Embarcacion.tpl`) en la zona del detalle añadimos el componente `CWLista` para la lista de puertos, que ahora no existía, en la que debemos añadir el parámetro `actualizaA`, ya que la lista de amarres va a depender del puerto elegido. En cambio, el amarre sí tiene un campo de texto para esa información, correspondiente al campo `lis_idamarre`, como queremos una lista, habrá que sustituir el `CWCampoTexto` por un `CWLista`.

```
{CWLista textoAsociado="Puerto" nombre="lis_puerto" actualizaA="lis_idamarre"
editable="true" size="40" visible="true" value=$defaultData_Estancias.lis_puerto
dataType=$dataType_Estancias.lis_puerto}
```

```
. . .
{CWCampoTexto textoAsociado="Idamarre" nombre="lis_idamarre" size="4" editable="true"
visible="true" value=$defaultData_Estancias.lis_idamarre
dataType=$dataType_Estancias.lis_idamarre}

sustituirlo por

{CWLista textoAsociado="Amarres" nombre="lis_idamarre" editable="true" size="40"
visible="true" value=$defaultData_Estancias.lis_idamarre
dataType=$dataType_Estancias.lis_idamarre}
```

Como no son listas estáticas como hemos definido en ejercicios anteriores, tenemos que definir las consultas que alimentarán ambas listas. Estas definiciones se hacen en el fichero `actions/principal/AppMainWindow.php`, siguiendo la estructura que veremos a continuación. La consulta debe devolver dos valores, etiquetados con alias “valor” y “descripcion”. Estos alias se definen para equiparar los valores de una lista en HTML, las opciones que componen una lista en HTML tienen la siguiente estructura:

```
<option value='1'>Jávea</option>
```

Por lo tanto “1” corresponderá a la etiqueta “valor”, y Jávea a la etiqueta “descripcion”.

Al método setList_DBSource() se le pasa una etiqueta ('PUERTOS' y 'AMARRES') que será con la que luego, en la clase manejadora, asociemos un componente lista a una definición.

Hay que tener en cuenta que la lista de puertos solamente nos deben aparecer aquellos que tienen amarres libres, y la lista de amarres, que dependerá del puerto elegido, solo mostrará los amarres libres en ese momento.

```
$sql = <<<query
       SELECT
              DISTINCT cpuerto as "valor",
              dpuerto as "descripcion"
       FROM
              puerto, amarre, estancia
       WHERE
              cpuerto = puerto_cpuerto
              AND amarre.idamarre = estancia.idamarre
              AND estancia.hasta < CURRENT_DATE
query;
$conf->setList_DBSource('PUERTOS',$sql);

$sql = <<<query
       SELECT
              amarre.idamarre as "valor",
              amarre.nombre as "descripcion"
       FROM
              amarre,estancia
       WHERE
              amarre.idamarre = estancia.idamarre
              AND estancia.hasta < CURRENT_DATE
query;
$conf->setList_DBSource('AMARRES',$sql);
```

Ahora nos queda definir ambas listas en la clase manejadora correspondiente al detalle (`actions/EmbarcacionEstancia/Estancias.php`)

En este caso no son listas estáticas, por lo tanto no añadimos las opciones aquí, sino que hacemos referencia a la etiqueta 'PUERTOS' que hemos creado antes en el fichero `AppMainWindow.php`

como definición de la lista, sí añadimos una opción sin contenido para que por defecto aparezca esa seleccionada y no un puerto.

La lista de AMARRES va a ser una lista dependiente del puerto elegido en la anterior, ello se indica con el método `setDependence()` donde le decimos qué campo de la plantilla es del que dependemos "`lisPuerto`" y con qué campo de la tabla corresponde "`puerto.cpuerto`".

Deberemos eliminar el tipo que nos ha generado Genaro automáticamente que usa "lis_idamarre" donde podemos visualizarlo en la siguiente tabla para poder incluir la definición de nuestro nuevo tipo y evitar duplicidades.

Eliminar:

```
$int = new gvHidraInteger(false, 4);
$this->addFieldType('lis_idamarre',$int);
```

Insertar:

```
$puerto = new gvHidraList('lis_puerto','PUERTOS');
$puerto->addOption('','');
$this->addList($puerto);

$amarre = new gvHidraList('lis_idamarre','AMARRES');
$amarre->addOption('','');
$amarre->setDependence(array("lis_puerto"), array("puerto_cpuerto"));
$this->addList($amarre);
```

- **Los campos fecha (desde y hasta) de las estancias deben llevar un calendario**

Este caso es sencillo, solamente indicar en la definición de la clase manejadora que queremos que el campo sea fecha y muestre el calendario.

```
$fecha = new gvHidraDate(false);
$fecha->setCalendar(true);
$this->addFieldType('lis_desde',$fecha);
$this->addFieldType('lis_hasta',$fecha);
```

Aquí vemos como queda el detalle con las listas dependientes y los calendarios en los campos desde y hasta:

Menu | Aplicación de ejemplo con gvHIDRA v.1.0.0 | INVITADO@BD-USUARIO | @-@ | 12:20 | 23/12/2014

Embarcacion

Codigo: 1
Matricula: VA-3-128-05
Nombre: Lébole
Manga: 1.6
Eslora: 3.2
Vela motor: m
Fecha compra: 12/12/2009
Caballaje: 10
Fecha baja:
Propietario dni: 1111111H

Reg. 01 de 04 | 01 02 03 | Ir a pág.

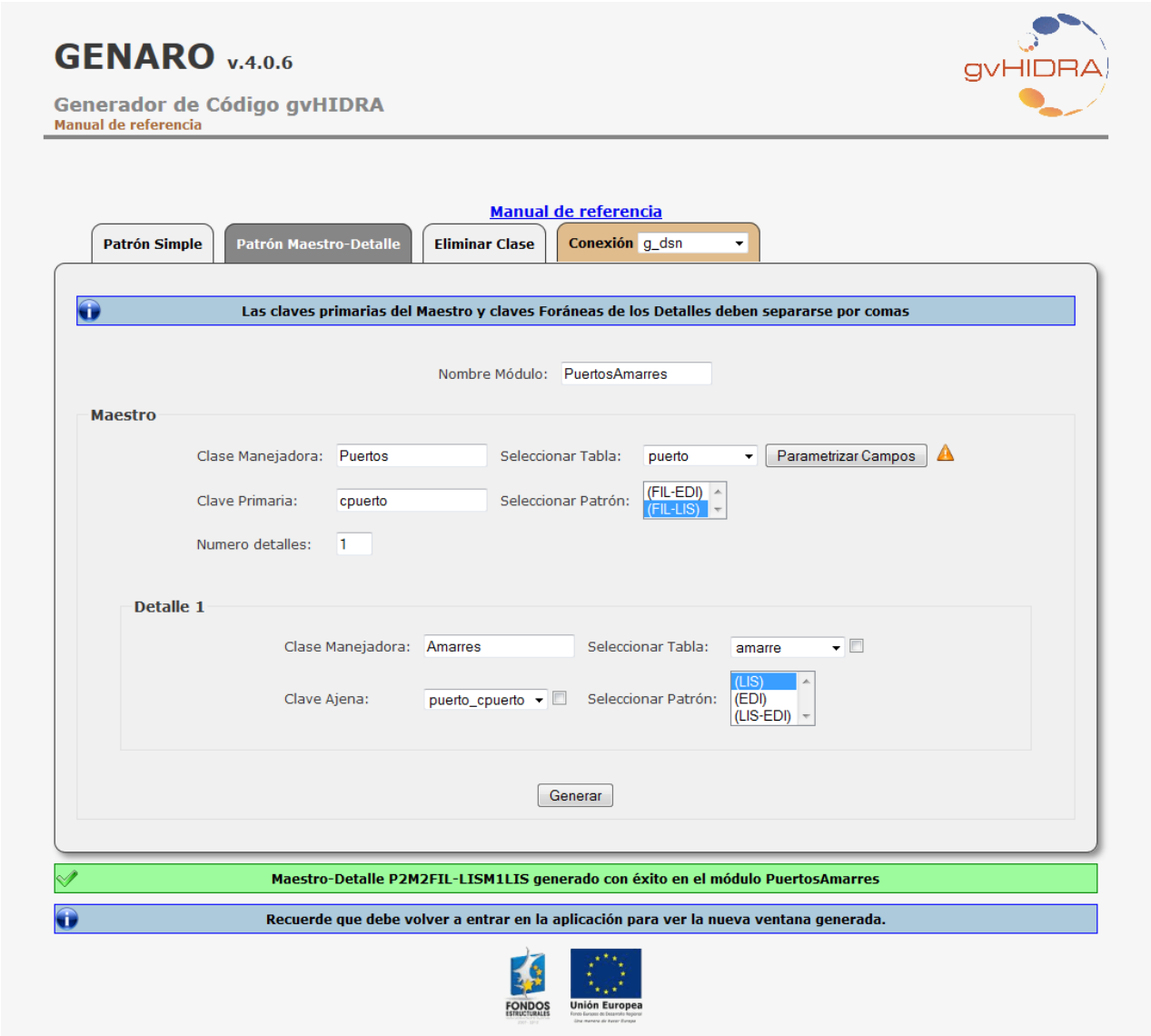
Estancias

Puerto	Amarres	Desde	Hasta
El port de Sagunt	PS-02A		
El portet de Dénia	D-01A		

(Nº reg. 0) Pg.. 01 de 01 | Ir a pág. | Guardar | Cancelar

MANTENIMIENTO DE PUERTOS Y AMARRES
Patrón de interfaz: Maestro-Detalle (FIL-LIS/LIS)
- Los datos del puerto no pueden modificarse desde esta pantalla, simplemente se muestran - Los datos de los amarres de un puerto si pueden modificarse. El estado de un amarre puede ser Correcto (c) En mantenimiento (m) y deshabilitado (d) para poder pasar a estados distintos del correcto, el amarre debe estar libre. - La anchura puede tener tres valores, grande (A), mediano (B) y pequeño (C) - El nombre de un amarre incluye al final la letra correspondiente a la anchura

Generamos la pantalla:



- **Los datos del puerto solo deben ser de lectura**

Este punto es igual que el primero del ejercicio anterior, por lo tanto solamente debemos cambiar el parámetro "`editable`" de todos los campos que se encuentran en el panel edición, `editable="false"`.

```
{CWCampoTexto textoAsociado="Cpuerto" nombre="lis_cpuerto" size="5" editable="false"
visible="true" value=$defaultData_Puertos.cpuerto  dataType=$dataType_Puertos.cpuerto}
{CWCampoTexto textoAsociado="Dpuerto" nombre="lis_dpuerto" size="30" editable="false"
visible="true" value=$defaultData_Puertos.lis_dpuerto  dataType=$dataType_Puertos.lis_dpuerto}
{CWCampoTexto textoAsociado="Direccion" nombre="lis_direccion" size="50" editable="false"
visible="true" value=$defaultData_Puertos.lis_direccion
dataType=$dataType_Puertos.lis_direccion}
{CWCampoTexto textoAsociado="Telefono" nombre="lis_telefono" size="20" editable="false"
visible="true" value=$defaultData_Puertos.lis_telefono
dataType=$dataType_Puertos.lis_telefono}
{CWCampoTexto textoAsociado="Municipio cmun" nombre="lis_municipio_cmun" size="5"
editable="false" visible="true" value=$defaultData_Puertos.lis_municipio_cmun
dataType=$dataType_Puertos.lis_municipio_cmun}
{CWCampoTexto textoAsociado="Num amarres" nombre="lis_num_amarres" size="4" editable="false"
visible="true" value=$defaultData_Puertos.lis_num_amarres
dataType=$dataType_Puertos.lis_num_amarres}
```

- **El estado de un amarre de un puerto puede ser Correcto (C), Mantenimiento (M) o Deshabilitado (D) Para poder pasar a estados distintos del Correcto el amarre debe estar libre.**

El estado del amarre lo vamos a definir como una lista estática con tres valores. Ya hemos hecho una antes para el estado de una embarcación, así que seguimos los mismos pasos.

Como dijimos, hay que definir el campo artificial en la consulta del panel, “lis_estado”, en la clase manejadora del detalle, `Amarre.php`:

```
$str_select = "SELECT idamarre as \"lis_idamarre\", nombre as \"lis_nombre\",
anchura as \"lis_anchura\", estado as \"lis_estado\", puerto_cpuerto
as \"lis_puerto_cpuerto\",
'' as \"lis_estado\"
FROM amarre";
```

Ahora añadiremos la definición de la lista:

```
$estado = new gvHidraList('lis_estado');
$estado->addOption('','');
$estado->addOption('C','Correcto');
$estado->addOption('M','Mantenimiento');
$estado->addOption('D','Deshabilitado');
$estado->setSelected('');
$this->addList($estado);
```

Por otra parte, hay que definir el componente lista en la plantilla. Por defecto Genaro nos lo ha creado como un campo de texto (`CWCampoTexto`), así que borraremos esa línea y la sustituiremos por una lista (`CWLista`).

```
{CWCampoTexto textoAsociado="Estado" nombre="lis_estado" size="15" editable="true"
value=$defaultData_Amarres.lis_estado dataType=$dataType_Amarres.lis_estado}

sustituirlo por:

{CWLista textoAsociado="Estado" nombre="lis_estado" size="30" editable="true"
visible="true" value=$dataType_Amarres.lis_estado
dataType=$dataType_Amarres.lis_estado}
```

Hasta aquí ya tenemos la lista desplegable, ahora nos queda realizar la acción de interfaz, es decir,

comprobar si el amarre está libre cuando se va a cambiar de estado distinto del Correcto.

Lo primero que hay que hacer es indicar en la plantilla (`p_Puertos.tpl`) que la lista tiene asociada una acción de interfaz, como en el ejemplo de Provincias y Municipios, hay que añadir el parámetro "`actualizaA`":

```
{CWLista textoAsociado="Estado" nombre="lis_estado" size="30" editable="true"
visible="true" value=$defaultData_Amarres.lis_estado dataType=$dataType_Amarres
.lis_estado actualizaA="lis_estado"}
```

Ahora hay que definir en la clase manejadora (`Amarres.php`) la acción de interfaz y su asociación al campo "`lis_estado`". En el constructor se asocia el evento al campo:

```
$this->addTriggerEvent('lis_estado','comprobarEstado');
```

Después hay que definir el método "`comprobarEstado`". El método realizará la comprobación del valor elegido en la lista, si es distinto de Correcto (C), hay que comprobar que el amarre está libre.

```
public function comprobarEstado($objDatos)
{
	$valor = $objDatos->getValue('lis_estado');
	if ($valor != 'C')
	{
		$idAmarre = $objDatos->getValue('lis_idamarre');
		$hoy = new gvHidraTimestamp();
		$this->getConnection()->prepararOperacion($hoy,TIPO_FECHA);
		$sql = <<<query
			SELECT
				*
			FROM
				estancia
			WHERE
				idamarre = $idAmarre
				AND hasta > $hoy
query;
		$res = $this->consultar($sql);
		if (count($res) > 0)
		{
			$this->showMessage('APL-3');
			return -1;
		}
	}
	return 0;
}
```

Como hemos dicho anteriormente, la definición de los mensajes, en este caso APL-3, se hace en el fichero `mensajes.php` que se encuentra en el raíz de la aplicación.

```
'APL-3'=>array('descCorta'=>'No puede cambiar el estado del amarre.','descLarga'=>'El
amarre está ocupado, por lo tanto no puede cambiar su estado.','tipo'=>'AVISO'),
```

- **La anchura puede tener tres valores, grande (A), mediano (B) y pequeño ©**

Aquí se nos pide otra lista, pero en este caso en vez de ser una lista desplegable haremos que sea una lista de radio buttons.

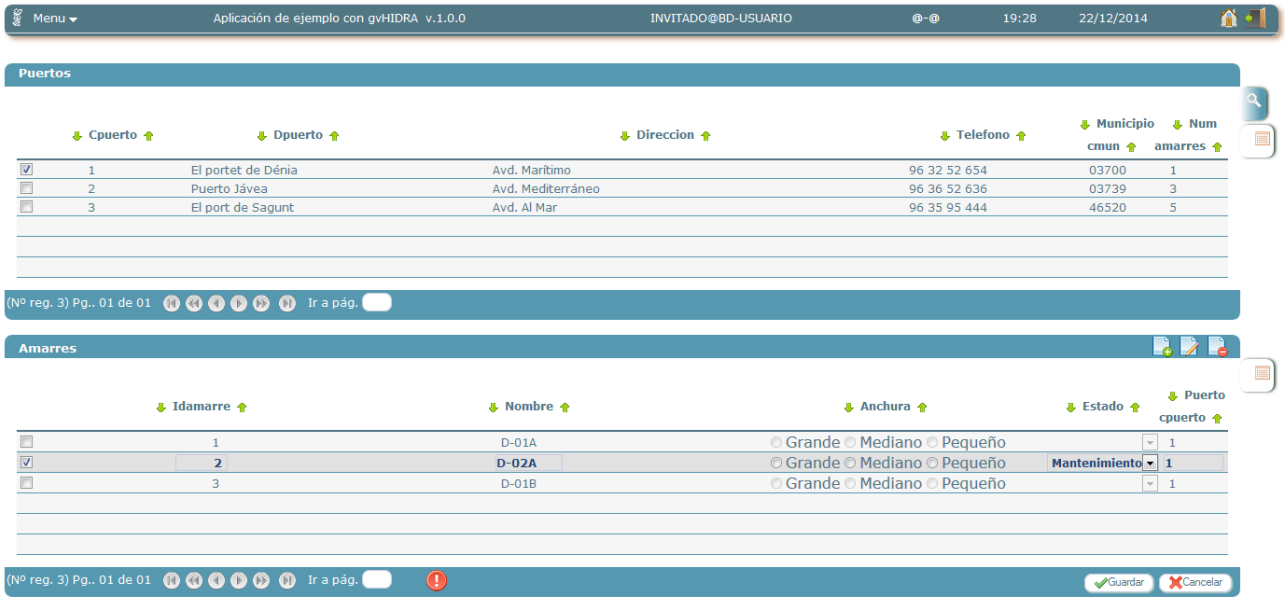
Para ello funcionamos casi igual que con una lista normal. La definición en la plantilla es igual:

```
{CWCampoTexto textoAsociado="Anchura" nombre="lis_anchura" size="10" editable="true"
value=$defaultData_Amarres.lis_anchura dataType=$dataType_Amarres.lis_anchura}

sustituirlo por:

{CWLista textoAsociado="Anchura" actualizaA="lis_anchura" nombre="lis_anchura"
size="10" editable="true" visible="true" value=$defaultData_Amarres.lis_anchura
dataType=$dataType_Amarres.lis_anchura}
```

Ahora nos queda la definición del tipo radio button en la clase manejadora, que se hace igual que una lista salvo que le indicamos el tipo de formato con el método `setRadio()`

```
$anchura = new gvHidraList('lis_anchura');
$anchura->addOption('A','Grande');
$anchura->addOption('B','Mediano');
$anchura->addOption('C','Pequeño');
$anchura->setRadio(true);
$this->addList($anchura);
```

A continuación vemos como queda el detalle:



- **El nombre de un amarre incluye al final la letra correspondiente a la anchura**

Esta modificación del nombre del amarre la realizaremos en el `preModificar()` y `preInsertar()`, así nos aseguramos de que antes de guardar en BD, se tiene ya el valor de la anchura y el del nombre.

La comprobación a realizar es la siguiente, válida para ambos métodos:

```
public function preModificar($objDatos)
{
        $datos = $objDatos->getAllTuplas();
        $nDatos = $datos;
```

```
        for($i=0;$i<count($datos);$i++)
        {
                $anchura = $datos[$i]['lis_anchura'];
                $nombre = $datos[$i]['lis_nombre'].$anchura;
                $nDatos[$i]['lis_nombre'] = $nombre;
        }
        $objDatos->setAllTuplas($nDatos);
        return 0;
}
```

4.6 Caso Práctico: Embarcaciones (II). Pantalla adaptativa.

4.6.1 Enunciado

Entrar en nuestra aplicación gvHidra y modificar una pantalla de tipo registro (EDI), el objetivo del ejercicio es aplicar el sistema de *grid* (columnas) a la pantalla para organizar el contenido en columnas.

Para realizar los cambios deberemos hacerlo desde la carpeta plantillas, y editar nuestro archivo ***.tpl***.

En este panel se usan inputs que tienen píxeles fijos, por lo tanto estos elementos no son *responsive*, una solución es aplicarle la siguiente clase **<div class="table-responsive"**> para que salga un *scroll* horizontal, de esta manera conseguiremos mantener la funcionalidad *responsive*. Este ***div*** encapsulará a todo el contenido que necesite de un *scroll*.

4.6.2 Solución

1. Descargar última versión de gvHidra.
2. Cambiar archivo gvHidraConfig.inc.xml: El archivo de configuración se ha ido modificando con nuevas versiones, por ello hay que usar el gvHidraConfig.inc.xml más reciente, para ello accedemos a la versión gvHidra descargada y cogemos el archivo gvHidraConfig.inc.xml de la carpeta appTemplate. Copiamos el archivo y lo reemplazamos por el de la aplicación (localizada en la carpeta raíz), incorporando previamente los datos de conexión.
3. Fijar el *custom* deseado[7], para ello, habrá que fijarnos en la etiqueta:

   <customDirName>*greyStyle***</customDirName>**

4. Reemplazar la carpeta "igep" por la descargada en el punto 1
5. Actualizar a la última versión del custom, normalmente basta con localizarlo en la caperta *appTemplate* de la versión descargada de gvHidra, copiar la carpeta custom y dejarla caer en la raiz de nuestro proyecto.
6. Adaptar la pantallas de la aplicación (ficheros .tpl)

   A) Cambiar imágenes: Las nuevas versiones de gvHidra incorpora el framework de Bootstrap, por lo tanto en las versiones más recientes se deja de usar imágenes por defecto (opcional) para utilizar los Glyphicons. Para llevar a cabo este cambio, basta con localizar en los fichero .tpl las subcadenas *imagen="*, este atributo visualiza una imagen que se identifica por un número. Por ejemplo la imagen se inserción sería: **imagen="01"** deberemos añadir un nuevo parámetro **iconCSS="glyphicon XXXX"**, donde XXXX sería alguno de los iconos disponibles en bootstrap:

   ```
   {CWBotonTooltip imagen="01" titulo="Insertar Embarcacion" funcion="insertar"
   actuaSobre="ficha" action="Embarcacion__nuevo"}
   ```

   Después:

[7] Las versiones anteriores funcionaban con el paquete de estilos "*blueStyle*", pero las nuevas versiones por defecto usan el paquete de estilos "*greyStyle*".

```
{CWBotonTooltip imagen="01" iconCSS="glyphicon glyphicon-plus" titulo="Insertar Embarcacion" funcion="insertar" actuaSobre="ficha" action="Embarcacion__nuevo"}
```

Las imágenes más habituales en la migración son las siguientes:

```
{CWBarra añadir... iconOut="glyphicon glyphicon-log-out" iconHome="glyphicon glyphicon-home" iconInfo="glyphicon glyphicon-info-sign"}
```

```
imagen="01" iconCSS="glyphicon glyphicon-plus"
```

```
imagen="02" iconCSS="glyphicon glyphicon-edit"
```

```
imagen="03" iconCSS="glyphicon glyphicon-minus"
```

```
imagen="04" iconCSS="glyphicon glyphicon-refresh"
```

```
imagen="13" iconCSS="glyphicon glyphicon-new-window"
```

```
imagen="50" iconCSS="glyphicon glyphicon-search"
```

```
imagen="41" iconCSS="glyphicon glyphicon-ok"
```

```
imagen="42" iconCSS="glyphicon glyphicon-remove"
```

```
{CWPaginador enlacesVisibles="3" iconCSS="true"}
```

Existen muchos más Glyphicons disponibles [getbootstrap.com/components/]

B) Añadir el parámero *smty_info*. El mismo debe incluirse en la cabecera de las plantillas (.tpl). Este parámetro nutre con información a los paneles de nuestra aplicación, no es obligatorio incorporarlo ya que la aplicación funciona correctamente sin este parámetro, pero es correcto incluirlo ya que al incluir este parámetro podremos visualizar más información.

```
{CWBarra usuario=$smty_usuario codigo=$smty_codigo customTitle=$smty_customTitle info=$smty_info modal=$smty_modal iconOut="glyphicon glyphicon-log-out" iconHome="glyphicon glyphicon-home" iconInfo="glyphicon glyphicon-info-sign"}
```

4.7 Caso Práctico: Embarcaciones (III). Multiidioma.

4.7.1 Enunciado

La versión 4.2 de gvHidra tiene soporte a la internacionalización (i18n) o multiidioma. Se propone al alumno realizar los cambios necesarios en la aplicación para que la aplicación pueda trabajar en castellano y valenciano, y al menos uno de los mantenimientos tenga soporte a ambos idiomas.

4.7.2 Solución

- Modificar el archivo gvHidraConfig.inc.xml. Deberemos añadir una etiqueta nueva para activar el multidioma. La etiqueta correspondiente es:

```
<LangZone default='esp' activate='true'>
    <lang id='esp' img='esp.png'>Español</lang>
    <lang id='val' img='val.png'>Valencià</lang>
</LangZone>
```

- Crear carpeta LANG: La carpeta estará situada en la carpeta raíz y contendrá los iconos de las banderas y los archivos de traducción. Habrá una bandera y un archivo de traducción por cada idioma que use la aplicación. Dentro del archivo de traducción las primeras etiquetas que visualizaremos serán exclusivas de gvHidra, estas etiquetas comienzan por : **gvhlang_**, **no se debe cambiar la descripción del nombre del** campo. Después ya trataremos las etiquetas personalizables de la aplicación.
- Crear las etiquetas personalizables en las TPL y en los ficheros de idioma. Para crear etiquetas de idiomas para paneles concretos deberemos primero incluir entre corchetes el nombre de la clase de la pantalla, por ejemplo:

```
[Embarcacion]
```

y después incluir las etiquetas de la clase, en el caso del archivo en castellano:

```
smty_tituloPanelFil = "Mantenimiento de Propietario"
smty_tituloPanelEdi = "Registro de Embarcación"
...
```

Una vez esten definidas las etiquetas deberemos ir a la tpl de nuestro panel y sustituir:

```
{CWBarraSupPanel titulo="Mantenimiento de Propietario"}
```

por nuestra etiqueta:

```
{CWBarraSupPanel titulo=#smty_tituloPanelFil#}
```

- Pantalla principal: Para traducir los tres bloques de la pantalla principal deberemos crear un archivo nuevo por cada idioma que tengamos de estos tres archivos:
  - menuAdministracion.xml
  - menuHerramientas.xml
  - menuModulos.xml

Los nombres de los nuevos archivos deberán tener la siguiente estructura:

menuModulos.val.xml

menuModulos.esp.xml

- Cambios en AppMainWindows: Para traducir la descripción de la pantalla principal tendremos que modificar el archivo citado. Existen dos casos a contemplar.
  - Se cambia de idioma pulsando una bandera en la barra superior de la pantalla principal.

```
$lang = $conf->getLanguage();
if($lang=='esp')
{
	$conf->setCustomTitle('Embarcaciones Castellano');
}
else
{
	$conf->setCustomTitle("Embarcaciones valencià");
}
```

  - Se modifica la función: **public function openApp($objDatos) { ... }**. Después de acceder a la aplicación mediante el login entraremos a esta función que recogerá el valor del idioma seleccionado en el login y que hará que la aplicación se visualice en el idioma anteriormente seleccionado.

```
public function openApp($objDatos)
{
	$conf = ConfigFramework::getConfig();
	if (isset($_REQUEST['lang']))
	{
		$lang = $_REQUEST['lang'];
	}
	else
	{
		$datosUsuario = IgepSession::dameDatosUsuario();
		$lang = $datosUsuario['language'];
	}

	$conf->setLanguage($lang);

	if($lang=='esp')
	{
		$conf->setCustomTitle('Embarcaciones Castellano');
```

```
        }
        else
        {
                $conf->setCustomTitle("Embarcaciones valencià");
        }
                return 0;
}
```

4.8 Caso Práctico: Embarcaciones (IV)

4.8.1 Enunciado

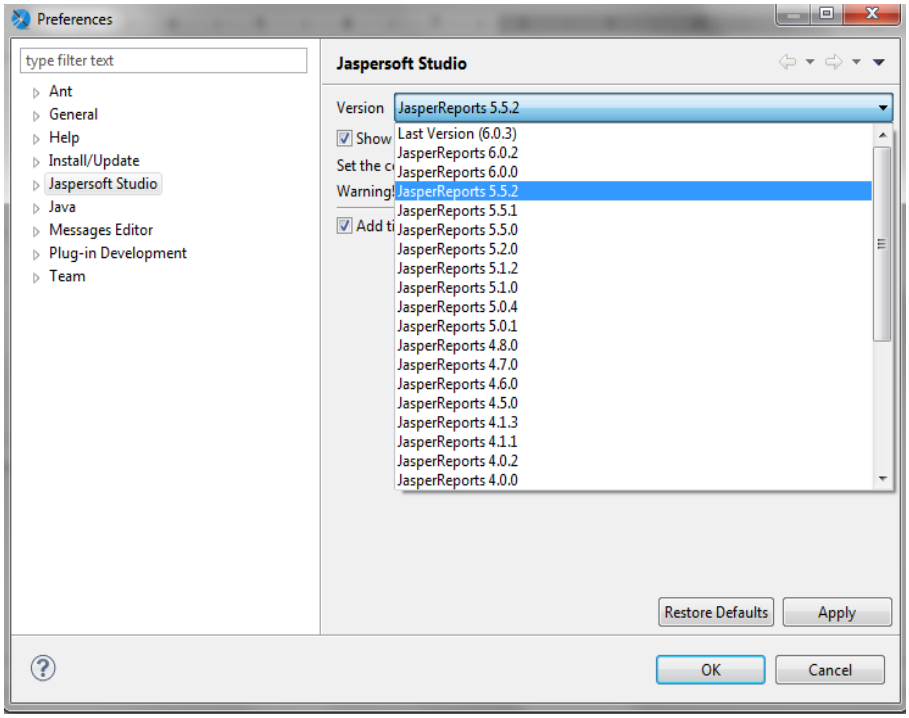
Tras poner en producción la primera versión de la aplicación, el usuario comienza a trabajar con la aplicación y detecta una serie de necesidades adicionales en la misma, concretamente, la inclusión de un informe o listado de propietarios de embarcaciones.

4.8.2 Solución

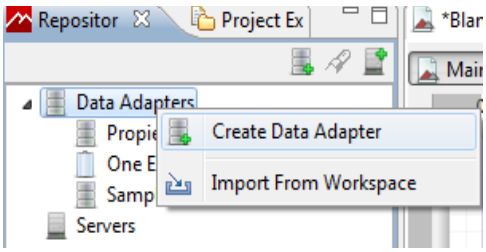
Para instalar el proyecto *jasper* en nuestro proyecto y en nuestro entorno de desarrollo local haremos uso de la herramienta Jaspersoft Studio.

Nos descargamos la herramienta y la instalamos.

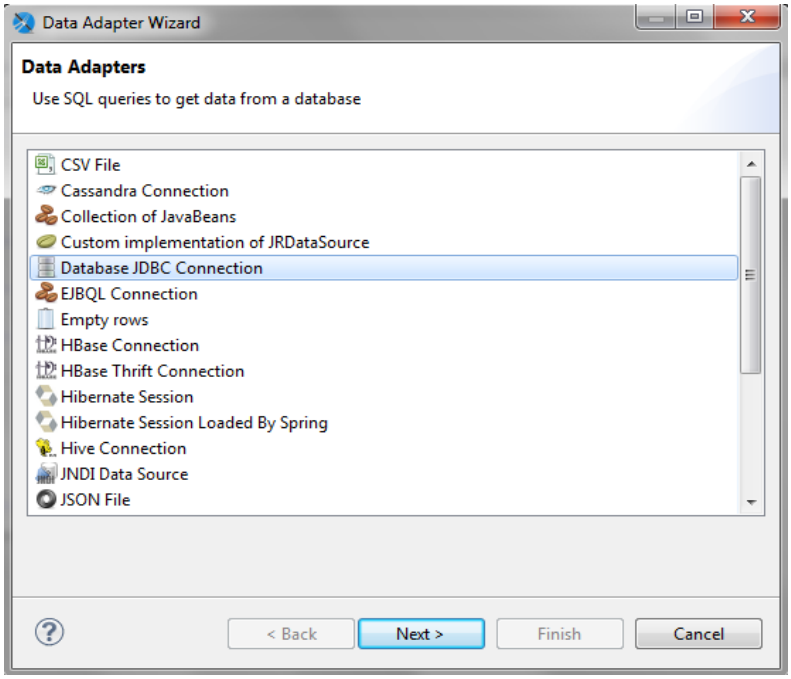
Una vez dentro de la aplicación lo primero que haremos será dirigirnos a Window/Preferences/Jaspersoft Studio y cambiar la versión de Jasper Resports.



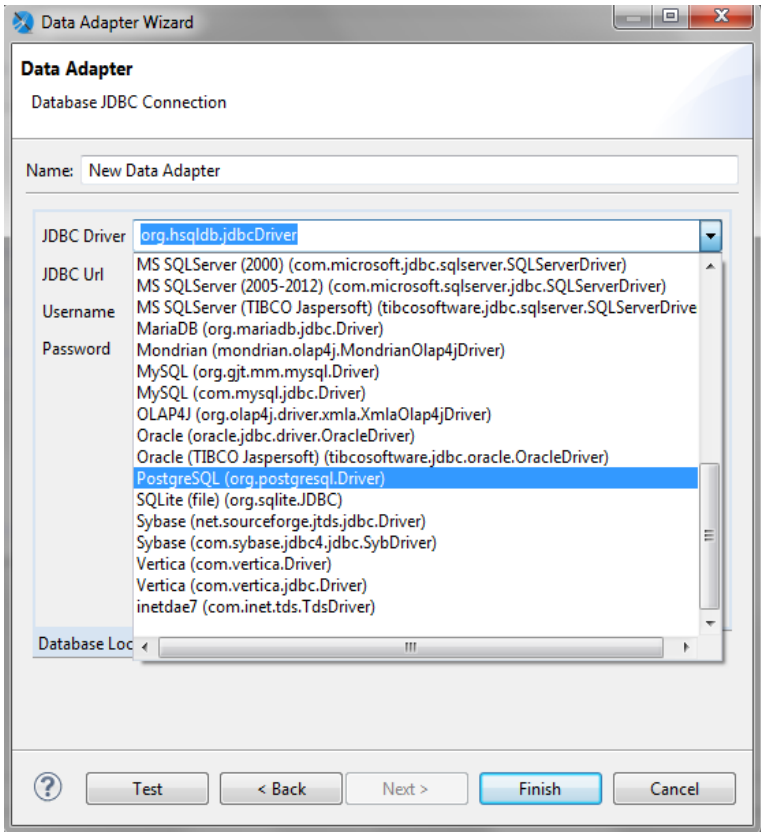
Antes de nada debemos conectarnos a una base de datos, para eso vamos a crear una conexión para PostgreSQL en la pestaña de la izquierda Repository Explorer pulsamos Data adapters y hacemos clic sobre el segundo botón del ratón para pulsar create data adapter.



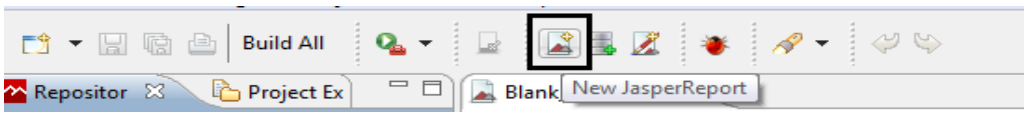
Escogemos la opción de Database JDBC Connection.



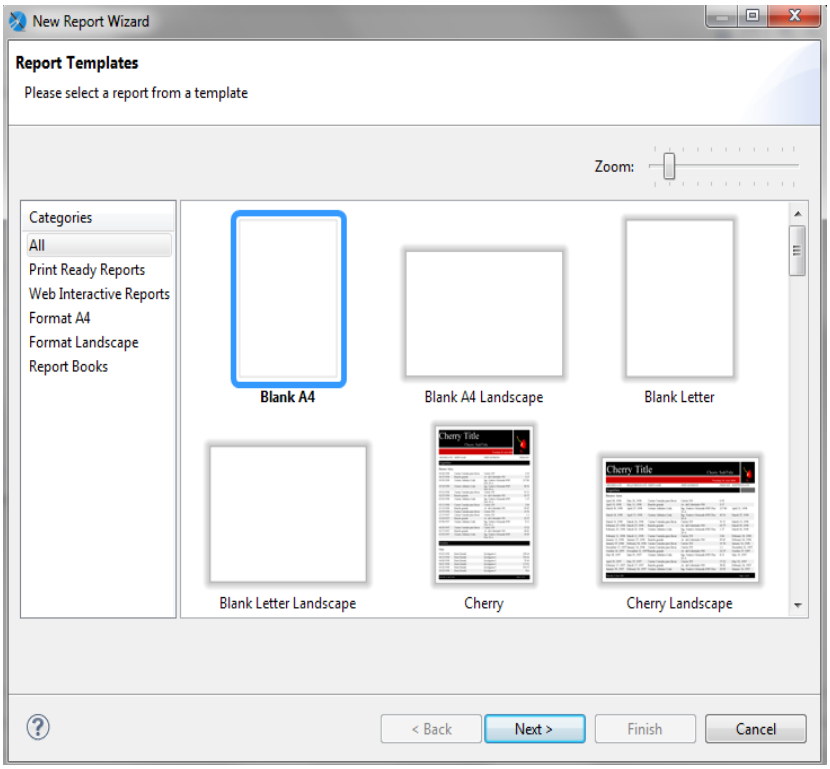
Visualizaremos un panel donde podremos escoger el tipo de base de datos que queramos usar, para nuestro ejemplo elegiremos PostgreSQL.



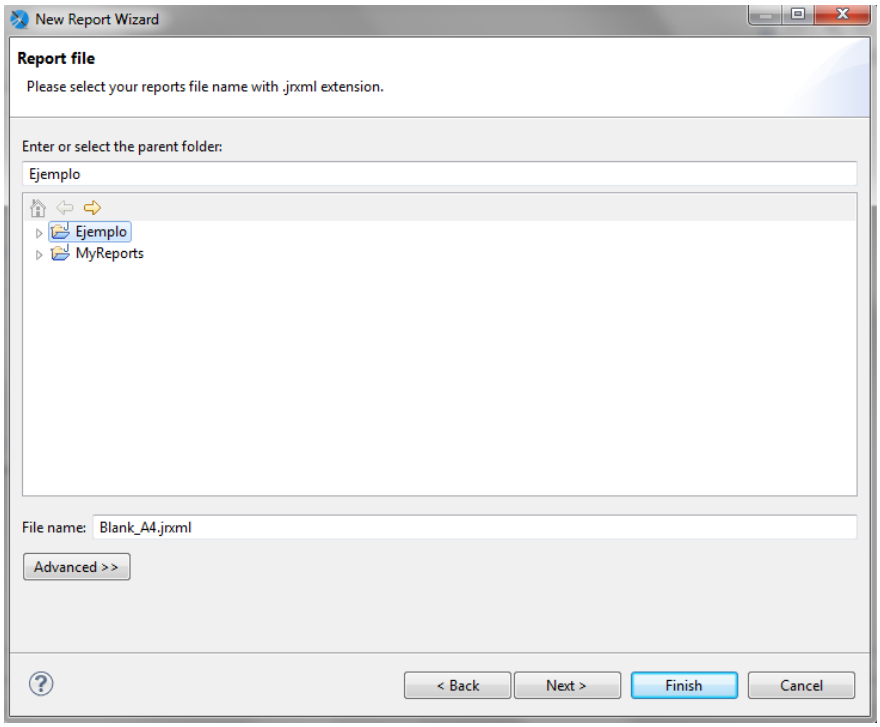
Escogemos nuestro tipo y rellenamos los datos correspondientes (nombre de la base de datos, usuario y contraseña). Realizada la conexión vamos a crear un nuevo report, para ello pulsamos el icono de la barra superior de nuestra aplicación NEW JasperReport.



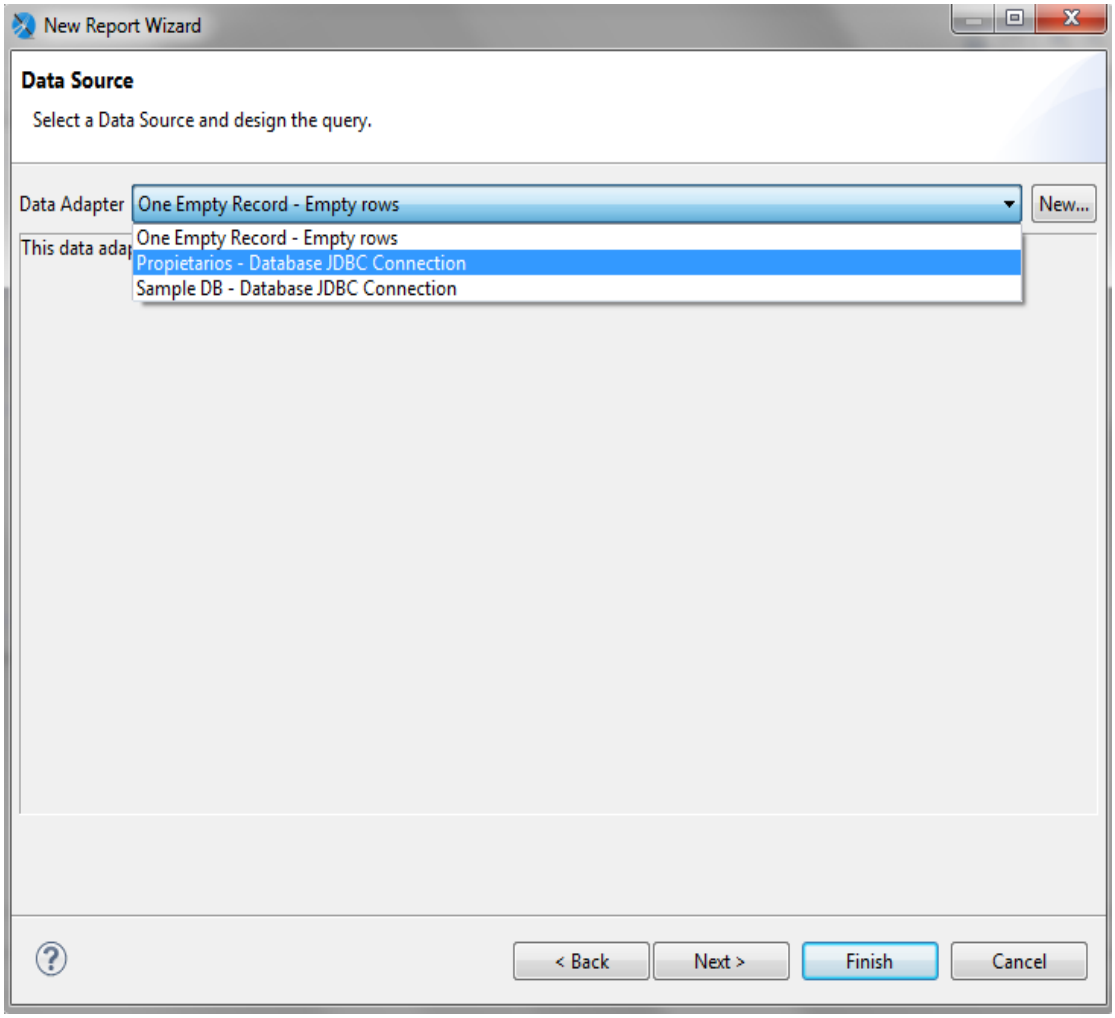
Escogemos una plantilla para nuestro report.



Pulsamos next y en la próxima pantalla deberemos elegir el proyecto donde queremos que se guarde nuestro report.

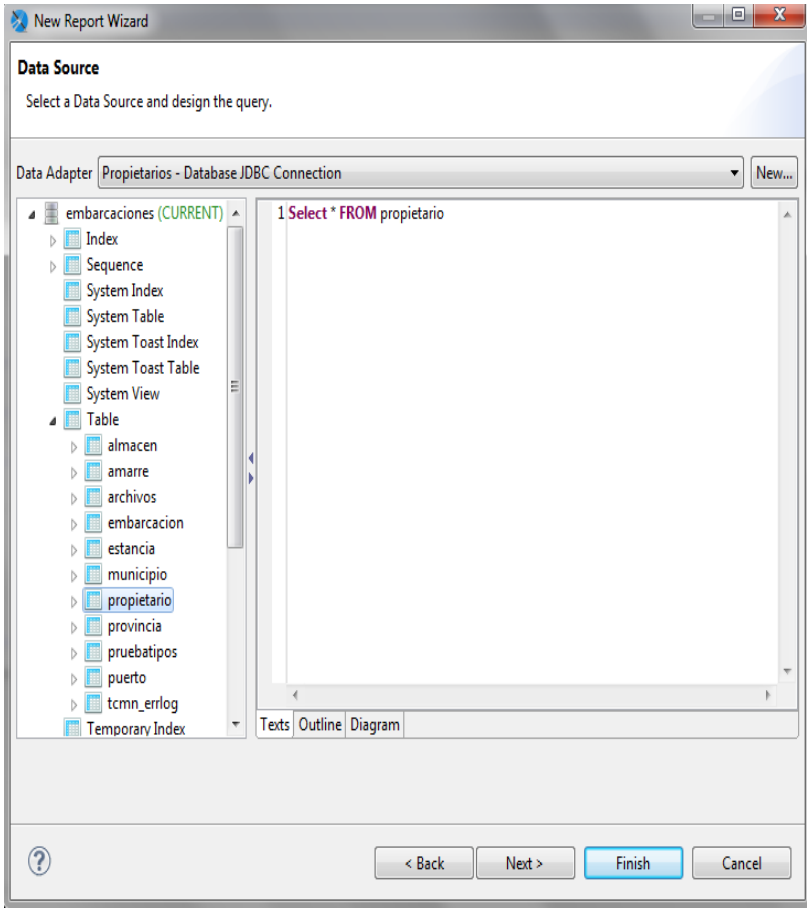


En la siguiente vista abriremos el listado y seleccionaremos nuestra conexión a base de datos creada en paso anteriores.

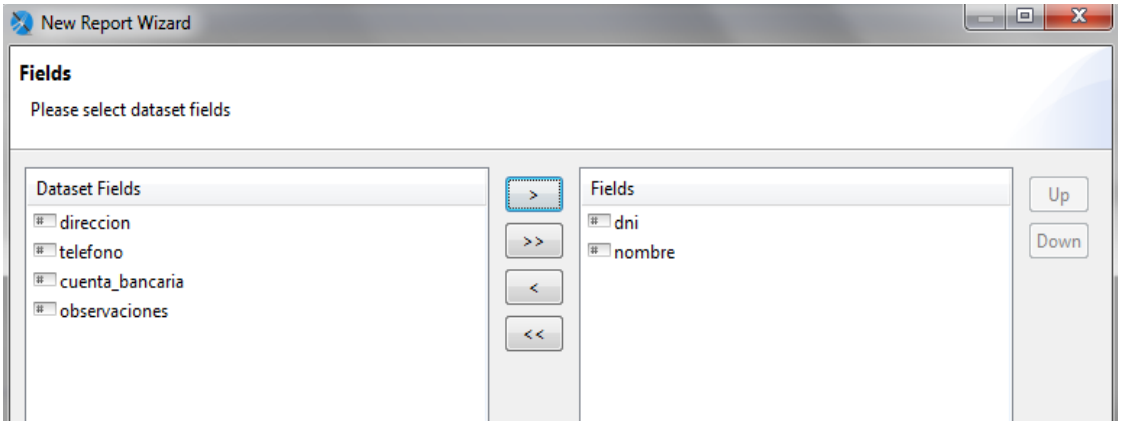


La ventana siguiente nos mostrará el contenido de nuestra base de datos, aquí deberemos escoger la información que queremos que se muestre en nuestro report, en nuestro ejemplo queremos que se visualice los campos DNI y Nombre de la tabla propietarios, para ello tenemos que introducir una sentencia SQL para seleccionar dicha tabla y le damos a next.

```
SELECT * FROM propietario
```



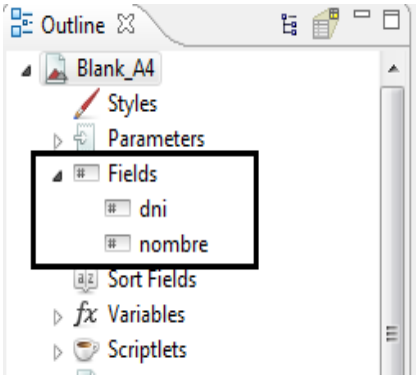
En la siguiente vista mostrará el resultado de nuestra sentencia SQL, donde deberemos seleccionar los campos DNI y Nombre y arrastrarlo a la derecha.



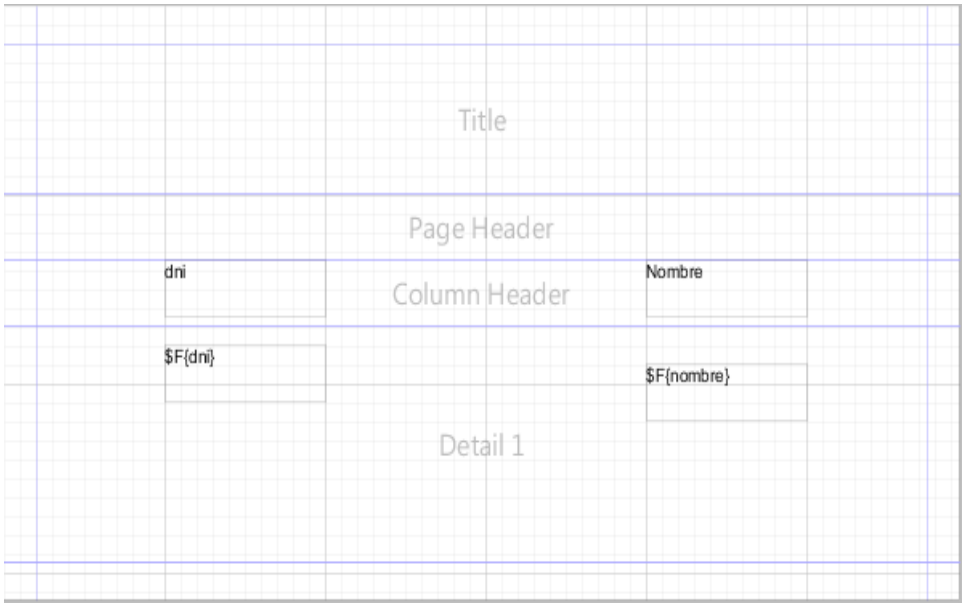
Después pulsamos al botón *finish* para acabar.

Ahora veremos que en nuestro proyecto se ha creado un archivo .jrxml, hacemos doble click sobre él y se abrirá una pantalla que tendremos que personalizar a nuestro gusto.

Para que se muestren los datos almacenados en la base de datos sobre el report deberemos arrastrar nuestros campos field a nuestro report.

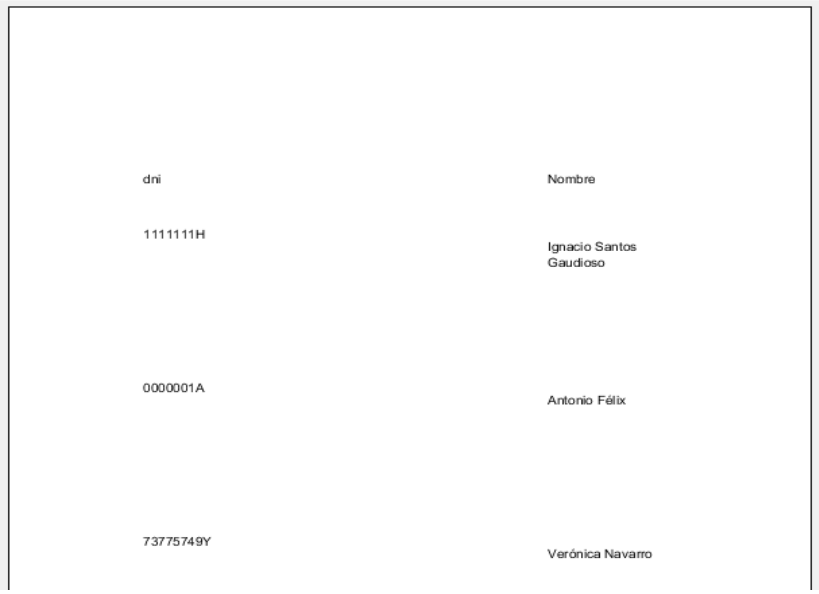


Una vez hecho esto quedaría así:



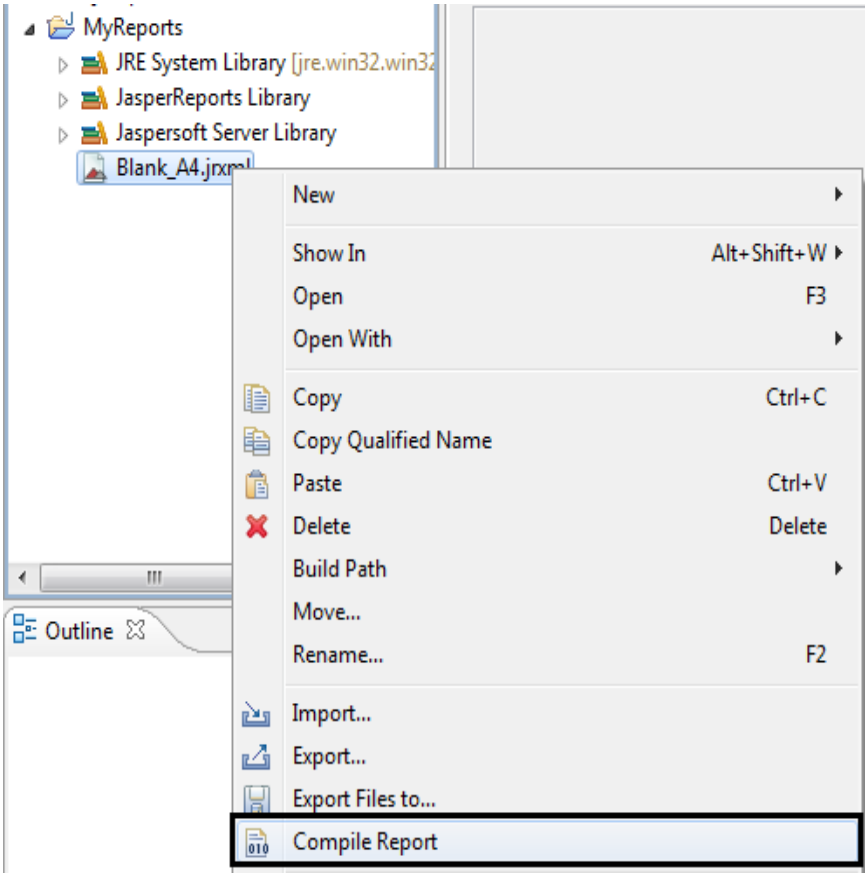
Como es un ejemplo se ha realizado un report lo más básico posible, siempre puede incluirse otros elementos como imágenes, textos fijos...

Para ver el resultado final debajo de nuestro report pulsamos en la pestaña preview.

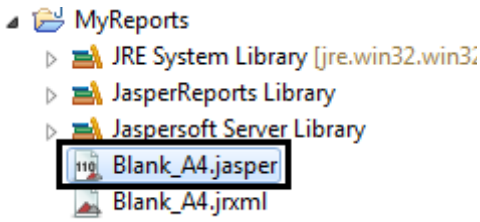


Vemos que se han leído bien los datos, ahora necesitamos compilar el report para generar el

archivo .jasper, para ello nos dirigimos a nuestro archivo generado .jrxml, clic al segundo botón y seleccionar Compile Report y a continuación se generará el archivo .jasper.



Vemos nuestro archivo .jasper generado.



Ya tenemos nuestro archivo.jasper creado, a partir de aquí vamos a explicar como enlazarlo dentro de una aplicación gvHidra.

Ahora entramos a nuestra aplicación de gvHidra y creamos una carpeta que se llame plantillasJasper, aquí dentro insertaremos nuestro archivo .jasper.

Una vez tengamos almacenado el archivo .jasper deberemos dirigirnos a nuestro proyecto gvhidra.

Lo primero que deberemos hacer es meternos a la URL http://www.gvpontis.gva.es/cast/gvhidra-herramienta/gvhidra-descargas/jasper/ y descargarnos el proyecto Jasper.

Una vez descargado nos dirigimos a la carpeta includes dentro de nuestro proyecto gvhidra y copiamos la carpeta Jasper dentro de ella.

Necesitaremos tener creado un panel en nuestra aplicación por lo tanto si no tenemos ningúno creado abrimos Genaro y lo generamos.

En este caso se ha creado un panel simple FIL-EDI llamado PostgresqlFILEDI

Vamos a trabajar sobre cuatro archivos.

Archivo plantilla/p_PostgresqlFILEDI.tpl

Comenzaremos modificando nuestro archivo tpl dentro de la carpeta plantillas.

Aquí insertaremos un botón nuevo llamado listar que al pulsarlo nos mostrará nuestro report, para ello creamos la action imprimir.

```
{CWBoton imagen="10" texto="Listar" class="button" accion="particular" action="imprimir"}
```

Archivo include/mappings.php

En la parte superior incluimos el modulo informeJasper.php

```
require_once('include/jasper/modulosPHP/informeJasper.php');
```

El método AdMapping se estructura de la siguiente manera:
AddMapping('claseManejadora__acción', 'claseManejadora');

El método AdForward se estructura de la siguiente manera:
AddForward('claseManejadora__acción', ' nombreForward ');

Relacionamos nuestra action con la clase manejadora.

```
$this->_AddMapping('PostgresqlFILEDI__imprimir', 'PostgresqlFILEDI');

$this->_AddForward('PostgresqlFILEDI__imprimir', 'gvHidraSuccess', 'index.php?
view=views/Postgresql/p_PostgresqlFILEDI.php&panel=buscar');

$this->_AddForward('PostgresqlFILEDI__imprimir', 'gvHidraPrint', 'index.php?
view=views/Postgresql/l_facturaJasper.php');
```

Archivo actions/PostgresqlFILEDI.php

Este archivo posee la clase manejadora.

En la parte superior incluimos el modulo informeJasper.php

```
require_once('include/jasper/modulosPHP/informeJasper.php');
```

Creamos dos variables públicas.

```
public $lanzarInforme;

//Este atributo será el informe Jasper
public $infJasper_demoEjemplo;
```

Dentro de la función accionesParticulares incluimos nuestra nueva acción.

```
public function accionesParticulares($str_accion, $objDatos) {

                if($str_accion == "imprimir")
                {
                        $objDatos->setOperation('seleccionar');


                        //Creamos el objeto que manejará el informe
                        $informeJ = new InformeJasper('DemoEjemplo');

                        //Especificamos la fuente de datos, en este caso, una BD relacional
                        $informeJ->setDataSourceType('sgbd');

                        //Especificamos los parámetros relativos a la conexión con la BD
relacional
                        //Si coinciden con los del DSN de IGEP, podemos importarlos con:
                        $conf = ConfigFramework::getConfig();
                        $g_dsn = $conf->getDSN('g_dsn');

                        $informeJ->importPearDSN($g_dsn);


                        //Fijamos el fichero jasper que nos hace de plantilla
                        $informeJ->setJasperFile('./plantillasJasper/Blank_A4.jasper');


                        $informeJ->addParam('DNI', $objDatos->getValue('DNI'), 'int');
                        $informeJ->addParam('Nombre', $objDatos->getValue('Nombre'), 'String');

                        //Asignamos el informe como variable de clase para poder
                        //acceder a él desde el fichero views donde realizaremos "la ejecución"
                        $this->infJasper_demoEjemplo = $informeJ;

                        //Hacemos un fork de la ejecución en una nueva ventana, recuperando el
forward del mapping
                        $actionForward=$objDatos->getForward('gvHidraPrint');
                        $this->openWindow($actionForward);


                        //Continua ejecución padre
                        $actionForward = $objDatos->getForward('gvHidraSuccess');
                        return $actionForward;
                }
    }
```

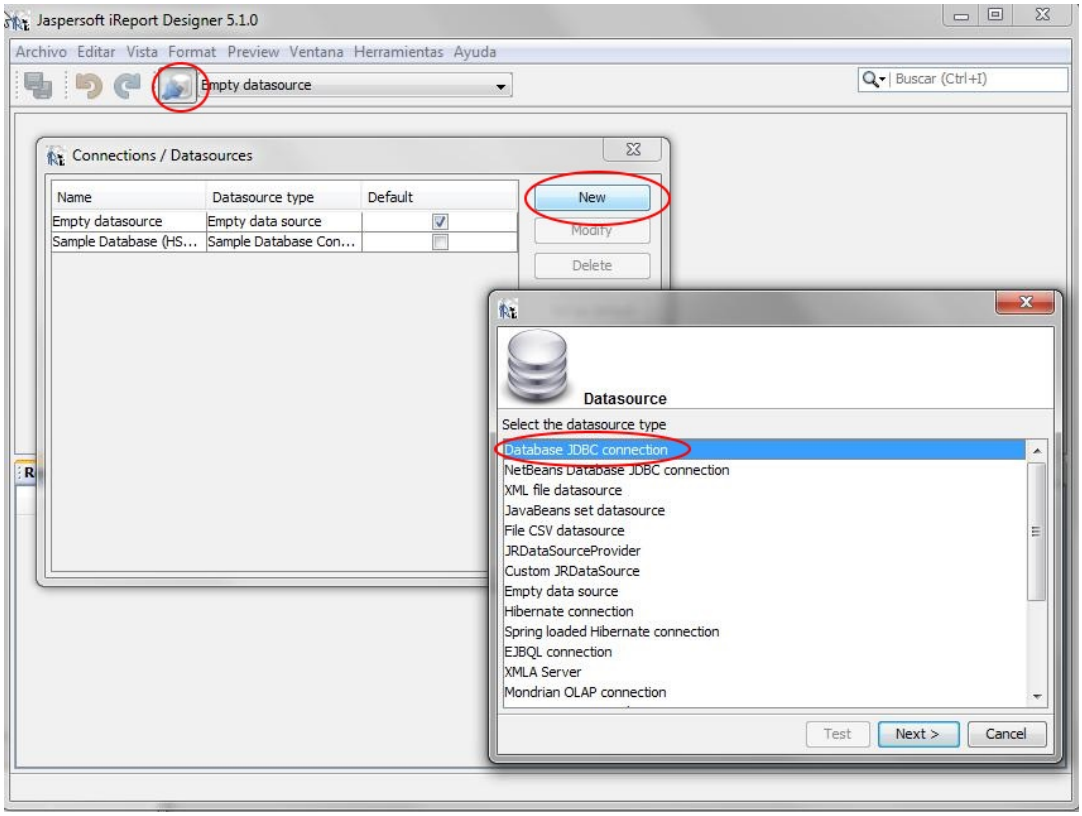
Archivo l_facturaJasper

Usamos las variables públicas creadas en la clase manejadora para generar nuestro report y visualizarlo en pdf.

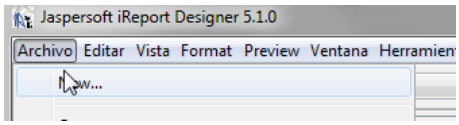
```
require_once('include/jasper/modulosPHP/informeJasper.php');

if (IgepSession::existeVariable("PostgresqlFILEDI","infJasper_demoEjemplo"))
{
        $informeJ = & IgepSession::dameVariable("PostgresqlFILEDI","infJasper_demoEjemplo");
        $informeJ->createResultFile('pdf');
}
IgepSession::borraVariable("PostgresqlFILEDI","lanzarInforme");
```

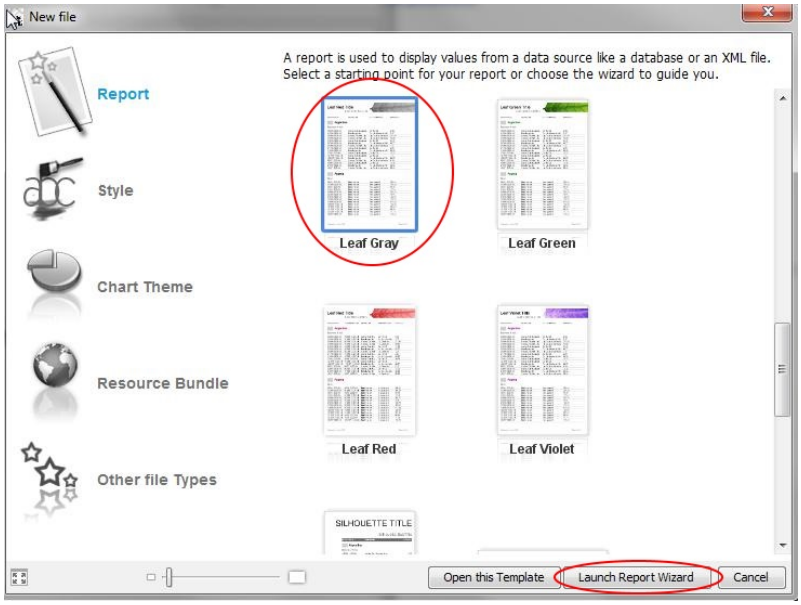
PARTE CON IREPORT

Necesitamos por instalar el proyecto *jasper* en nuestro proyecto y en nuestro entorno de desarrollo

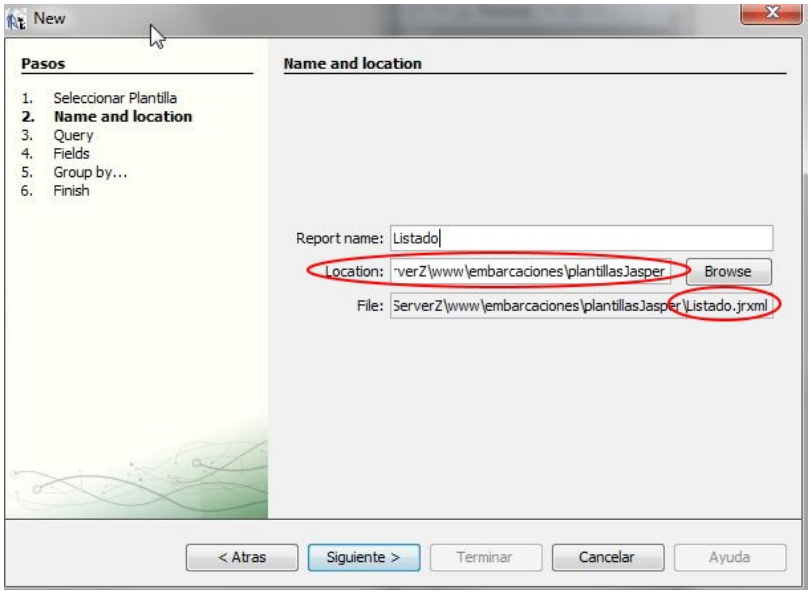


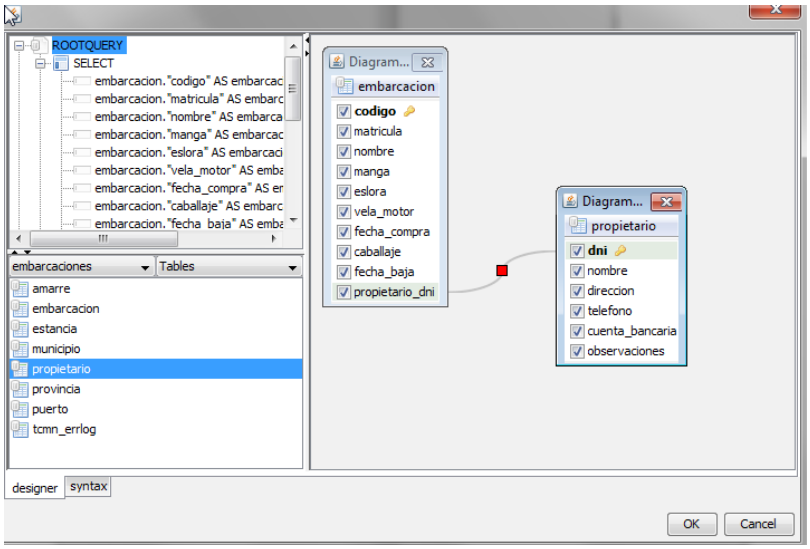
Tras la instalación y testeo de que la conexión funciona de forma correcta, nos podemos servir del asistente (*wizard*) para crear el listado.



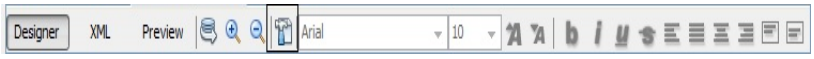


Dentro nuestro proyecto gvhidra crearemos una carpeta en el directorio raíz llamada plantillasJasper, en esta carpeta se almacenará los archivos generados .jrxml y .jasper.

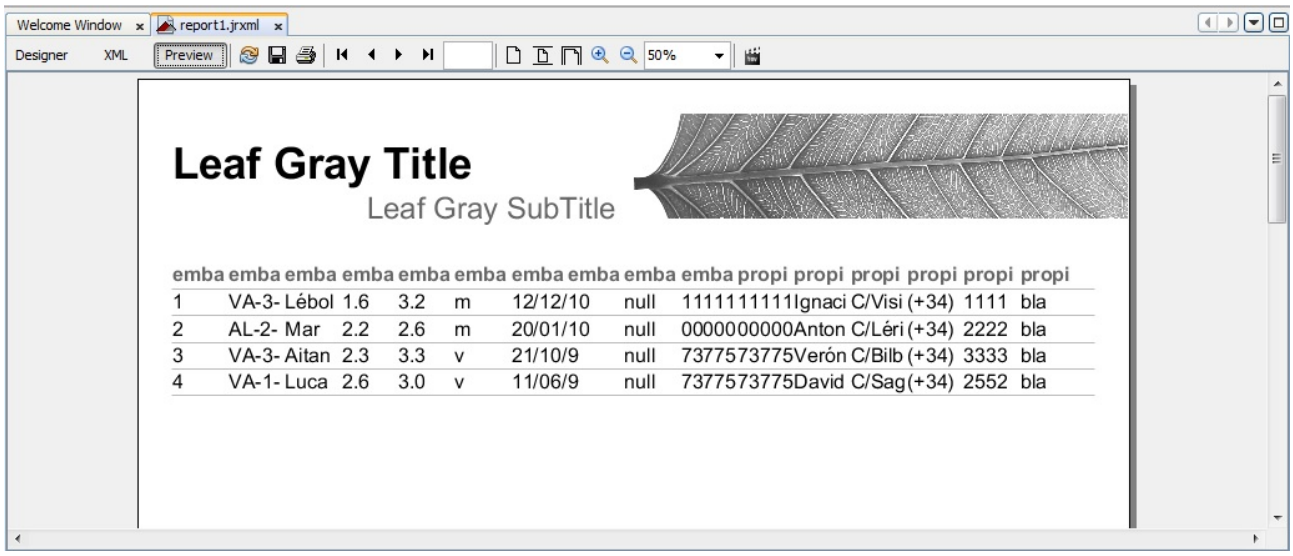




Después de acabar estos pasos deberemos compilar nuestro nuevo archivo generado, deberemos seleccionar el siguiente botón o presionar Preview:



Después de la compilación visualizaremos una pantalla como esta:



Y se nos habrá generado el archivo .jasper en la carpeta que hemos indicado en pasos anteriores, en nuestro caso en plantillasJasper.

Una vez tengamos almacenado el archivo .jasper deberemos dirigirnos a nuestro proyecto gvhidra.

Lo primero que deberemos hacer es meternos a la URL http://www.gvpontis.gva.es/cast/gvhidra-herramienta/gvhidra-descargas/jasper/ y descargarnos el proyecto Jasper.

Una vez descargado nos dirigimos a la carpeta includes dentro de nuestro proyecto gvhidra y copiamos la carpeta Jasper dentro de ella.

Necesitaremos una tabla (FIL) por lo tanto si no tenemos ninguna creada abrimos Genaro y la generamos.