[MANUAL TÉCNICO ETAB]

Tablero de control electrónico

Dashboard

Esta herramienta está pensada para ofrecer una visión general del progreso de la Iniciativa hacia sus resultados esperados.

ING. ELIECER RAMIREZ ESQUINCA
[ramirez.esquinca@gmail.com]

Tabla de contenido

1. INTRODUCCIÓN

El sistema de información eTAB es parte de la iniciativa Salud Mesoamérica 2015 (SM2015). Esta es una iniciativa cuyo fin es reducir las inequidades en salud que están afectando al 20% más pobre de la población en Centro América y México. Esta iniciativa también tiene como objetivo apoyar los esfuerzos de los gobiernos de la región para alcanzar los Objetivos del Milenio.

Salud Mesoamérica 2015, pone especial atención a las áreas de salud reproductiva, nutrición maternal y neonatal maternal inmunización, y la prevención y control del dengue y la malaria. Para este fin Salud Mesoamérica 2015trabajara en conjunto con los ministerios de salud de la región y el Sistema de Salud Publica Mesoamericano. Este proyecto es parte de la plataforma de integración regional conocido como Proyecto Mesoamérica.

Los resultados esperados que incluyen una reducción significante en las tasa de mortalidad infantil para niños de menos de cinco años. Esta iniciativa también esta busca reducir la malnutrición crónica en la niñez y las mujeres embarazadas. Estos cambios son críticos para mejorar las estadísticas sobre partos y para ofrecer mejores condiciones para el crecimiento del recién nacido. A su vez esta iniciativa busca tener un efecto directo en comunidades pobres sobre la cobertura y calidad de vacunas, control pre y post natal y acceso a planificación familiar entre otros servicios.

Salud Mesoamérica 2015 espera generar conocimiento de relevancia global sobre cómo aumentar asistencia en salud de bajo costo en comunidades pobres. Para este fin el sistema de información eTAB permite analizar y dar seguimiento a los indicadores en salud con los que trabaja este proyecto.

El presente manual técnico describe cada uno de los componentes y tecnologías utilizadas en el sistema eTAB y los pasos necesarios para su instalación y adecuación.

2. OBJETIVO

El siguiente documento va dirigido a los administradores de sistemas o área de tecnologías de la información y tiene por objeto guiar al o los participantes, a conocer la estructura de la creación y forma de operación del eTAB explicando las tecnologías utilizadas y la metodología en la programación.

3. ALCANCE

En el presente documento describe la información mínima de cada uno de los componentes y/o tecnologías utilizadas en este proyecto.

4. TECNOLOGÍAS UTILIZADAS

El Tablero eTAB es un servicio Web disponible para que dependencias del sistema de salud suban sus datos para poder analizarlos, generar gráficas y reportes.

La aplicación cuenta con un módulo para efectuar la extracción, transformación y carga de datos (ETL) desde diferentes fuentes. Estos datos son agregados y almacenados en una base de datos relacional (OLTP). Los datos están organizados por catálogos de referencia e Indicadores medibles. Los usuarios del sistema pueden administrar estos indicadores y catálogos y todos sus tributos usando el las herramientas que brinda el sistema. Para efectuar consultas en línea los datos son agregados dentro de tablas optimizadas para el análisis en línea (OLAP). Las tablas para análisis son actualizadas periódicamente usando procedimientos almacenados de PostgresSQL.

La gestión de consultas a las tablas de análisis OLAP se realiza por medio de un servidor dedicado. La interacción entre el servidor OLAP y el resto de la aplicación se realiza por medio de consultas AJAX. El resultado de las consultas al servidor OLAP, es procesado usando JQuery y graficado usando la librería de gráficos D3.

Todo el software utilizado para creación del eTAB son paquetes de software libre. Estos incluyen:

- PostgreSQL: Gestor de bases de Datos
- Apache: Servidor de páginas web
- PHP: Lenguaje de desarrollo de la Aplicación eTAB
- GitHub: Gestor de control de versiones de código fuente
- RabbitMQ: Servidor de Mensajería
- JQuery: Lenguaje para interfaces de usuario
- Bootstrap: Framework para interfaces de usuario
- D3.js: Librería para la generación de gráficos
- Symfony: Framework y Entorno de desarrollo para PHP

4.1 POSTGRESQL.

Versión 9.1

Actualmente el sistema únicamente puede utilizar PostgreSQL por la siguiente razón: La aplicación debe proveer la capacidad de analizar datos para cualquier indicador. Cada indicador está construido con varios datos y relacionados por una formula almacenada en el sistema. Es posible crear una tabla por cada grupo de datos, con la limitante de que es necesario conocer el dato antes de guardarlo, lo cual no es sostenible a futuro.

La base de datos necesita guardar datos sin conocer de antemano sus características. Esto se logra usando un esquema de datos genérico EAV (entidad-atributo-valor). El manejo de esquemas EAV es implementado de diferentes formas para diferentes gestores de bases de datos, el Tablero eTAB usa la implementación de PostgreSQL la cual crea un tipo especial de dato llamado HSTORE que sirve para almacenar conjuntos de pares clave / valor dentro de un único valor PostgreSQL. Esto puede ser útil en diversos escenarios, como filas con muchos atributos, que rara vez se examinan, o los datos semi-estructurados. Teclas y los valores son simplemente cadenas de texto.

4.2 APACHE

Versión 2.2

El servidor HTTP Apache es un servidor web HTTP de código abierto, para plataformas Unix (BSD, GNU/Linux, etc.), Microsoft Windows, Macintosh y otras, que implementa el protocolo HTTP/1.1[2] y la noción de sitio virtual.

4.3 PHP

Versión 5.4

Es un lenguaje de programación de uso general de código del lado del servidor originalmente diseñado para el desarrollo web de contenido dinámico. Fue uno de los primeros lenguajes de programación del lado del servidor que se podían incorporar directamente en el documento HTML en lugar de llamar a un archivo externo que procese los datos. El código es interpretado por un servidor web con un módulo de procesador de PHP que genera la página Web resultante. PHP ha evolucionado por lo que ahora incluye también una interfaz de línea de comandos que puede ser usada en aplicaciones gráficas independientes. Puede ser usado en la mayoría de los servidores web al igual que en casi todos los sistemas operativos y plataformas sin ningún costo.

El lenguaje que se ha utilizado dentro de una estructura de desarrollo MVC manejada Symfony versión 2.4 El cual por su extensa documentación se explica en el capítulo siguiente.

4.4 GITHUB

Es una plataforma para alojar proyectos utilizando el sistema de control de versiones Git. Utiliza el framework Ruby on Rails por *GitHub, Inc.* (anteriormente conocida como *Logical Awesome*).

4.5 RABBITMQ

Es un software de negociación de mensajes de código abierto, y entra dentro de la categoría de middleware de mensajería. Implementa el estándar Advanced Message Queuing Protocol (AMQP). El servidor RabbitMQ está escrito en Erlang y utiliza el *framework* Open Telecom Platform (OTP) para construir sus capacidades de ejecución distribuida y conmutación ante errores.

La carga de datos se apoya de las librerías de este paquete para crear una 'lista de espera' para evitar que el servidor se sature al recibir demasiadas peticiones simultaneas.

4.6 JQUERY

Es una biblioteca de JavaScript, que permite simplificar la manera de interactuar con los documentos HTML, manipular el árbol DOM, manejar eventos, desarrollar animaciones y agregar interacción con la técnica AJAX a páginas web.

4.7 BOOTSTRAP

Bootstrap es un framework que hace HTML, CSS y Javascript simple y flexible para componentes e interacciones de interfaz de usuarios populares.

4.8 D3

Antes conocida como Protovis, D3 es una biblioteca de JavaScript para manipular documentos basados en datos. D3 ayuda dar vida a los datos usando HTML, SVG y CSS. D3 enfatiza los estándares Web ofreciendo todas las capacidades de los navegadores modernos sin ligarse a una estructura propietaria. A diferencia de otras librerías, D3 no crea imágenes, sino que interactúa la página para crear los gráficos usando elementos de HTML5 como Canvas y SVG.

4.9 SYMFONY

Es un completo framework diseñado para optimizar el desarrollo de las aplicaciones web basado en el patrón Modelo Vista Controlador. Para empezar, separa la lógica de negocio, la lógica de servidor y la presentación de la aplicación web. Proporciona varias herramientas y clases encaminadas a reducir el tiempo de desarrollo de una aplicación web compleja. Además, automatiza las tareas más comunes, permitiendo al desarrollador dedicarse por completo a los aspectos específicos de cada aplicación. El resultado de todas estas ventajas es que no se debe reinventar la rueda cada vez que se crea una nueva aplicación web.

5. REQUERIMIENTOS PARA LA INSTALACIÓN DEL ETAB

El sistema eTAB está basado en las arquitecturas cliente/servidor. Los requerimientos del sistema son pues diferentes según el tipo de función (cliente o servidor) para la cual el dispositivo se vaya a destinar. También dependen del tamaño de las bibliotecas, archivo o centro de documentación en la que el sistema se va a instalar.
El eTAB es un sistema que tiene una alta escalabilidad. Esto significa que el sistema puede operar en forma básica en equipos de prestaciones modestas y que a partir de allí es posible migrar a instalaciones superiores sin pérdida de información ni de infraestructura.
Los requerimientos mínimos y las características recomendadas para servidores y clientes se detallan a continuación. Es importante comprender que las características mínimas que se especifican para los servidores corresponden a instalaciones pequeñas y que no debe pretenderse que esta configuración mínima sea apta para bibliotecas de mediano tamaño que manejan varias decenas o centenas de miles de información documentos, ejemplares y/o usuarios.

5.1 REQUERIMIENTOS DE SOFTWARE

Software básico	Características mínimas	Características recomendadas
Sistema Operativo	Debian 6, Ubunto 10	Debian 7, Ubuntu 14
Manejador de Base de Datos	PostgresSQL	
Servidor de HTTP	Apache 2	Apache 2 o mayor
PHP	Php 5.3.8	Php 5.4 o mayor

5.2 REQUERIMIENTOS DE HARDWARE

Dispositivo	**Características mínimas**	**Características recomendadas**
Procesador	Procesador de 1.00 Ghz con opción de multiprocesamiento simétrico de al menos dos procesadores.	Procesador 4.00 Ghz con opción de multiprocesamiento simétrico de al menos cuatro procesadores.
Memoria RAM	2 gb	8 gb
Memoria Cache:	6.0 MB FSB 1066 Mhz	6.0 MB FSB 2.80 GHz
Disco Duro	500 gb	1 Tb

6. INSTALACIÓN.

Es muy importante poner atención al indicador "#" significa que el comando debe ser ejecutado como usuario root y "$" que debe ser ejecutado como un usuario normal, en ambos casos desde una consola de comandos.

6.1 INSTALACIÓN DE LOS REQUERIMIENTOS DESDE UN SERVIDOR DEBÍAN

Actualizamos la lista de paquetes del sistema operativo.
apt-get update
Instalamos todas las librerías y aplicaciones que se utilizan en el sistema (PHP, PostgreSQL y Git).

apt-get install php5 php5-pgsql php5-sqlite sqlite php5-xdebug php-apc php5-cli php5-xsl php5-intl php5-mcrypt apache2 postgresql acl git-core curl postgresql-contrib php5-ldap php5-mysql php5-sybase php5-json

6.2 CREAR USUARIO Y DIRECTORIO DE TRABAJO

El directorio y usuario a utilizar pueden variar de acuerdo a los que se deseen elegir en cada

instalación. Como ejemplo se usará un usuario llamado **etab** y el directorio de instalación **/var/www/ etab**
Creamos el usuario
adduser etab
Creamos el directorio
mkdir /var/www/ etab

Asignamos como dueño del directorio al usuario que acabamos de crear, en nuestro caso es etab
chown etab : etab /var/www/ etab

Nos cambiamos al usuario que es dueño del directorio etab
su etab

Accedemos a la carpeta web del Apache
$ cd /var/www

6.3 OBTENER EL CÓDIGO FUENTE

Puedes descargarlo desde: https://github.com/checherman/etab o clonar el repositorio
$ git clone https://github.com/checherman/etab etab

Recuerda que actualmente estamos en el directorio */var/www* y el último parámetro del git clone es la carpeta en donde se descargará el código fuente del repositorio, en este nuestro caso es etab.

NOTA: A partir de este punto todos los comandos se deben ejecutar dentro de la carpeta en que se ha descargado el código fuente.

Si se desea tener las aportaciones del equipo SM2015 Chiapas, es necesario cambiar de rama el repositorio, esto se logra ejecutando la siguiente sentencia:

git checkout Chiapas

6.4 INSTALAR COMPOSER

Composer es una librería de PHP para el manejo de dependencias. Para instalarlo, dentro de la carpeta donde descargaste el código fuente se debe ejecutar:
$ curl -s https://getcomposer.org/installer | php

6.5 INSTALAR TODAS LAS LIBRERÍAS NECESARIAS

$ php composer.phar install

Dado que Symfony2 es un proyecto Open Source, depende de librerías y paquetes de terceros, por lo que puede presentarse el caso que al ejecutar composer install se produzca un error de dependencias. Al ejecutar el composer install este lee el archivo composer.json donde se encuentran las dependencias del proyecto. Al terminar la instalación este crea el archivo composer.lock el cual contiene las especificación exacta de las versiones de los paquetes instalados, por lo que puede utilizar como alternativa el siguiente archivo composer.lock (Anexo No 1) e intentar nuevamente la instalación, este solo en caso de presentarse el problema de dependencias.

Durante la instalación se solicitarán los siguientes parámetros (Si desea conservar el valor por defecto para cada entrada es suficiente con presionar Enter para confirmar el valor):
database_driver: pdo_pgsql
Esta variable contiene el driver php que manejará la comunicación con la base de datos en la capa de acceso a datos, dado que la plataforma trabaja con PostgreSQL su valor por defecto será pdo_pgsql, se recomienda no modificar este valor.

database_host: localhost
Se refiere al host donde se encuentra alojado el servidor de base de datos, en nuestro caso es el mismo que el servidor web.

database_port: null
El puerto del manejador de base de datos, en nuestro caso es null ya que utiliza el puerto por defecto para el PostgreSQL (5432).

database_name: indicadores
Nombre de la base de datos, más adelante se creará la base de datos con ayuda de Symfony.

database_user: admin
Nombre del usuario para la base de datos, este se creará más adelante en la sección Configuración de PostgreSQL.

database_password: admin
Contraseña del usuario para la base de datos.

mailer_transport: smtp
Protocolo para la transferencia de correo electrónico.

mailer_host: localhost
Servidor de correo electrónico.

mailer_user: null
Usuario para el servidor de correo electrónico.

mailer_password: null
Contraseña del usuario para el servidor de correo electrónico.

locale: es_MX
Lenguaje por defecto para la aplicación.

secret: 295125e6c66ab2a1038b62ad3c910733510
Esta es una cadena que debe ser única, se utiliza para la generación de las tokens CSRF, pero que podría ser utilizado en cualquier otro contexto en donde una cadena única es útil, como por ejemplo, la encriptación de las contraseñas de usuario.

archivo_vitacora: %kernel.logs_dir%/minsal.log
Archivo donde se guardará el registro de eventos de la aplicación. %kernel.logs_dir% es una variable de Symfony2 que hace referencia a la ruta relativa app/logs/ (con respecto a directorio de instalación).

carpeta_etab_mondrian: %kernel.root_dir%/mondrian/
Carpeta de esquemas generados por la aplicación para Pentaho, esta variable es utilizada por Saiku. %kernel.root_dir% es una variable de Symfony2 que hace referencia a la ruta relativa app/ (con respecto a directorio de instalación).

conexion_bd_pentaho: Minsal
Nombre de conexión a base de datos dentro de Pentaho.

listado_metadata: datasources.etab
Archivo que lista esquemas existentes y su conexión. Este archivo debe estar ligado al fichero ../pentaho-solutions/system/olap/datasources.xml

7. CONFIGURACIÓN SERVIDOR WEB

Esto es para una instalación de prueba en una máquina local, la instalación real en un servidor el administrador de servicios deberá realizar esta configuración con los parámetros más adecuados: ip, dominio, configuración en el DNS, etc.

7.1 CONFIGURAR UN VIRTUALHOST

Creamos el archivo para la definición del VirtualHost
nano /etc/apache2/sites-available/etab.salud.sm

El contenido será similar a esto:

```
<VirtualHost 127.0.0.7>

   ServerName etab.salud.sm
   DocumentRoot /var/www/etab/web

   <Directory /var/www/etab/web >
      Options Indexes FollowSymLinks MultiViews
      AllowOverride All
```

```
        Order allow,deny
        allow from all
    </Directory>

    ErrorLog ${APACHE_LOG_DIR}/etab-error.localhost.log
    # Possible values include: debug, info, notice, warn, error, crit,
    # alert, emerg.
    LogLevel warn
    CustomLog ${APACHE_LOG_DIR}/etab-access.localhost.log combined

</VirtualHost>
```

En el archivo /etc/hosts agregamos la línea
127.0.0.7 etab.salud.sm

Habilitamos el VirtualHost
a2ensite etab.localhost

También es recomendable activar el módulo mod_rewrite
a2enmod rewrite

Reiniciar apache
/etc/init.d/apache2 restart

7.2 PERMISOS SOBRE CARPETAS

Es necesario tener soporte para ACL en la partición en que está el proyecto y luego ejecutar
$ setfacl -R -m u:www-data:rwx -m u:`whoami`:rwx app/cache app/logs web/uploads

$ setfacl -dR -m u:www-data:rwx -m u:`whoami`:rwx app/cache app/logs web/uploads

7.3 VERIFICAR LA CONFIGURACIÓN

Entra a la siguiente dirección desde el navegador *http://* etab.salud.sm */config.php*. Si aparece algún error debe ser corregido antes de continuar.

7.4. CONFIGURACIÓN DE POSTGRESQL

Editar archivo de configuración

Como usuario root realizar:

1. Editar el archivo /etc/postgresql/9.1/main/pg_hba.conf
2. Cambiar la siguiente línea, sustituir la última palabra por md5

local all all md5

Reiniciar PostgreSQL

/etc/init.d/postgresql restart

7.5 CREAR EL USUARIO DUEÑO DE LA BASE DE DATOS

Se creará el usuario dueño de la base de datos, las opciones utilizadas dependerán de los criterios que se quieran seguir, se muestra un ejemplo, ejecutar createuser --help para la explicación de las opciones. El nombre utilizado y la clave deben corresponder con los parámetros especificados al ejecutar php composer.phar install en unas secciones anteriores

su postgres

$ createuser -d -s -R -P admin;

Al finalizar presionar la combinación Ctrl+D 2 veces para regresar al usuario etab y continuar con la instalación.

7.6 CREAR LA BASE DE DATOS

Symfony hace uso del bundle Doctrine para el manejo de la capa de datos.
$ app/console doctrine:database:create

Este comando creará la base de datos.
$ app/console doctrine:schema:update --force

Este comando creará la estructura de las tablas del sistema.

7.7 CARGAR DATOS INICIALES

$ app/console doctrine:fixtures:load

Con este comando se insertan los datos iniciales del sistema.

7.8 CREAR UN USUARIO ADMINISTRADOR DEL ETAB

Symfony hace uso del bundle FOSUser para la administración de usuarios.
$ app/console fos:user:create --super-admin

Con este *usuario* se iniciará **sesión** en la aplicación web al terminar instalación.

7.9 INSTALACIÓN DE HSTORE

HStore es un tipo especial de campo de PostgreSQL.

- Conectarse al servidor de base de datos con el usuario postgres, esto se logra ejecutando desde una ventana de comando las siguientes sentencias:

psql -U postgres -d database_name

El parametro database_name es el que se estableció anteriormente al ejecutar composer install.
Ahora procedemos a crear la extensión hstore:
create extension hstore;

Salir de la línea de comandos del PostgreSQL con \q.

- Crear la tabla especial que no se manejará con el ORM, hacerlo con el usuario dueño de la base de datos (no con el usuario postgres, a menos que este mismo sea el dueño de la base de datos).

psql -U database_user -d database_name

El parametro database_user y database_name se establecieron anteriormente al ejecutar composer install.

```
CREATE TABLE fila_origen_dato(
   id_origen_dato integer,
   datos hstore,
   ultima_lectura timestamp,
```

```
  FOREIGN KEY (id_origen_dato) REFERENCES origen_datos(id) on update CASCADE on delete CASCADE
);
```

Salir de la línea de comandos del PostgreSQL con \q.

Si se prefiere, hay una alternativa de interfaz gráfica para la administración de PostgreSQL, este es *pgAdmin*. Para instalar este administrador ejecutar la siguiente sentencia:
aptitude install pgadmin3

7.10 INSTALACIÓN DE RABBITMQ

RabbitMQ es un sistema de mensajería empresarial completo y altamente confiable basado en el estándar AMQP. En este proyecto será utilizado para la carga masiva de datos y así evitar cuelgues o saturación del servidor.

- Agregar el repositorio

sh -c 'echo "deb http://www.rabbitmq.com/debian/ testing main" >> /etc/apt/sources.list'

- Agregar la clave pública

wget http://www.rabbitmq.com/rabbitmq-signing-key-public.asc

apt-key add rabbitmq-signing-key-public.asc

- Ejecutar

apt-get update

- Instalar el paquete

apt-get install rabbitmq-server

- Verificar que el servicio de rabbitmq esté corriendo

/etc/init.d/rabbitmq-server start

- Habilitar la interfaz web de administración

rabbitmq-plugins enable rabbitmq_management

/etc/init.d/rabbitmq-server restart

- Cargar la interfaz web: entrar a la dirección *http://localhost:55672/mgmt/*. El usuario por defecto es **guest** y la clave **guest**
- Iniciar las colas

$ src/MINSAL/IndicadoresBundle/Util/iniciar_colas.sh

Pueden aparecer mensajes de aviso como "/usr/bin/nohup: redirecting stderr to stdout" solo debemos presionar ENTER

- Además es necesario configurar el **CRON** para que ejecute periódicamente la carga de datos, con esto se llamará al proceso origen-dato:cargar que verificará para cada indicador si le corresponde realizar la carga de datos según se haya configurado: diario, mensual, bimensual, trimestral, cuatrimestral, semestral o anual. Un ejemplo podría ser crear el archivo: /etc/cron.d/carga-php-siig con el siguiente contenido:

#Ejecutar cada dia a las 00:00

0 0 * * * www-data test -x /usr/bin/php && /usr/bin/php /var/www/siig/app/console origen-dato:cargar

7.11 CARGAR LA APLICACIÓN

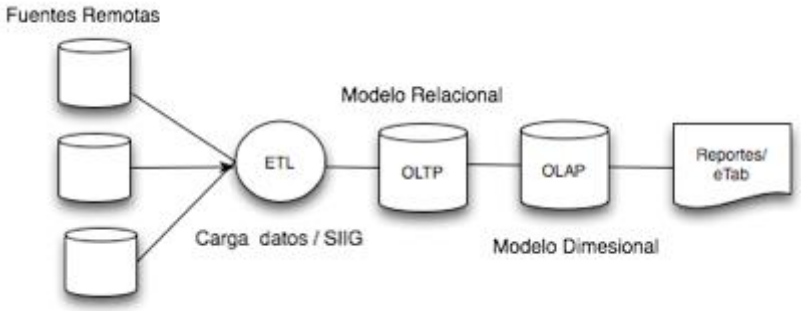
En este punto estamos listos para cargar la aplicación desde:

http://etab.salud.sm

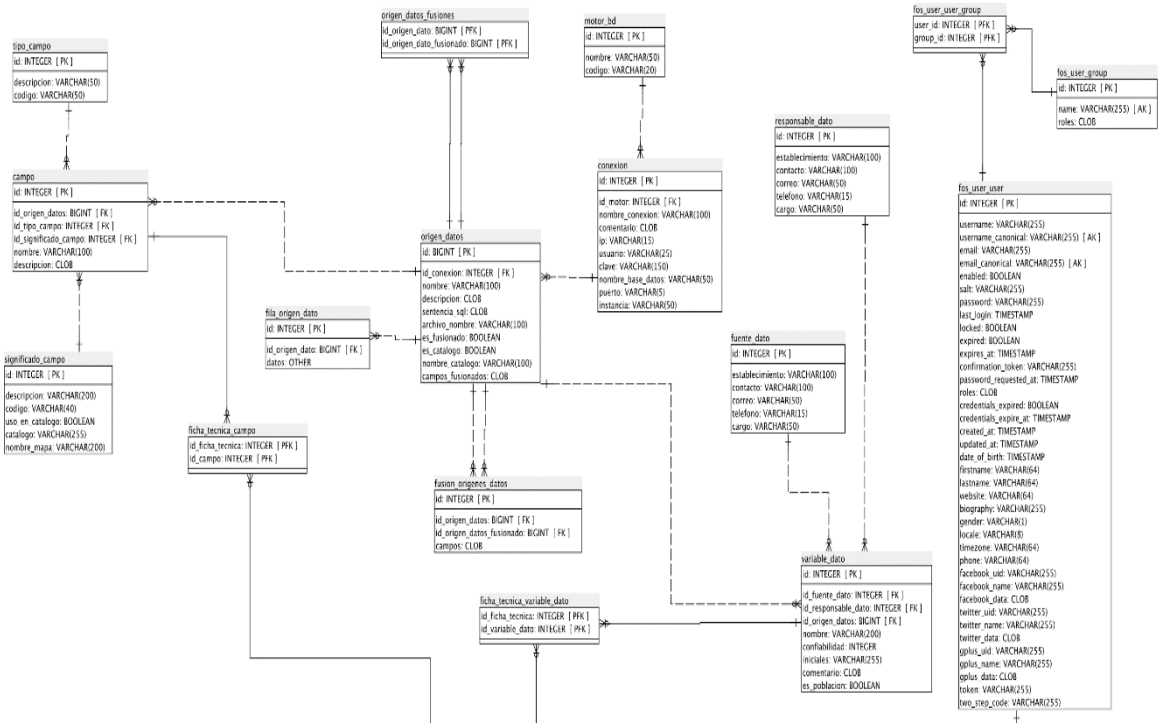
8. MODELO DE DATOS

Los datos que maneja el sistema provienen de distintas fuentes y son de una naturaleza tal que es necesario utilizar el modelo de base datos sin esquema/genérico EAV. La tabla EAV (Fila_origen_dato) y demás tablas auxiliares son parte del almacenamiento de datos transaccional (OLTP) de la aplicación. Esto facilita el manejo de datos de cualquier indicador sin importar sus propiedades. Los cubos de análisis multidimensional (OLAP) son generados usando estos valores genéricos y están descritos en la sección de Gestión de Cubos OLAP. Las tablas de los cubos OLAP usan un esquema de estrella mientras que las tablas del almacenamiento OLTP usan un modelo relacional. El Siguiente Diccionario de Datos y Diagrama ER describen la estructura del almacenamiento transaccional (OLTP) de la Aplicación.

8.1 ESQUEMA GENERAL DE LA APLICACIÓN



8.2 DIAGRAMA ENTIDAD RELACIÓN



9. SYMFONY PARA PROGRAMADORES

Definición: *« Symfony es un framework PHP de tipo full-stack construido con varios componentes independientes creados por el proyecto Symfony »*

9.1 PRINCIPALES CARACTERÍSTICAS

- Su código, y el de todos los componentes y librerías que incluye, se publican bajo la licencia MIT de **software libre**.
- La **documentación** del proyecto también es libre e incluye varios libros y decenas de tutoriales específicos.
- Aprender a programar con Symfony te permite acceder a una gran **variedad** de proyectos: el *framework* Symfony2 para crear aplicaciones complejas, el*micro framework* Silex para sitios web sencillos y los componentes Symfony para otras aplicaciones PHP.
- Según GitHub, Symfony es el proyecto PHP más **activo**, lo que garantiza que nunca te quedarás *atrapado* en un proyecto sin actividad. Además, el líder del proyecto, Fabien Potencier, es la segunda persona más activa del mundo en GitHub .

- Aunque en su desarrollo participan cientos de programadores de todo el mundo, las **decisiones** técnicas importantes siempre las toma Fabien Potencier, líder del proyecto. Esto evita el peligro de que surjan *forks*absurdos y la comunidad se fragmente.
- Los **componentes** de Symfony son tan útiles y están tan probados, que proyectos tan gigantescos como Drupal 8 están construidos con ellos.
- En todo el mundo se celebran varias **conferencias** dedicadas exclusivamente a Symfony. Para que te hagas una idea del tamaño de la comunidad, la conferencia Symfony española (llamada deSymfony) es el evento PHP más grande del país.

10. SYMFONY PARA ADMINISTRADORES DE SISTEMAS

Definición: *« Symfony es un conjunto de librerías que se utilizan para crear aplicaciones PHP »*

10.1 PRINCIPALES CARACTERÍSTICAS

- Las versiones actuales de Symfony requieren disponer de **PHP 5.3.8** o superior. Así evitas instalar en tus servidores versiones PHP peligrosas llenas de problemas de seguridad y a la vez no es un requisito técnico demasiado exigente.
- En producción, las aplicaciones Symfony solamente necesitan **permiso de escritura** en dos directorios internos de la propia aplicación. Además, Symfony incluye varias herramientas gráficas y de consola para depurar fácilmente los errores que se produzcan en las aplicaciones.

- Para evitar el uso de **contraseñas** en archivos de configuración, Symfony permite establecer los parámetros de configuración de las aplicaciones a través de variables de entorno del propio servidor.

- La **seguridad** es tan importante para el proyecto Symfony, que antes de su lanzamiento, se encargó una auditoría de seguridad a una empresa independiente.
- La herramienta Capifony (basada en el proyecto Capistrano y creada por miembros de la comunidad Symfony) simplifica el **deploy** de las aplicaciones Symfony, incluso en múltiples servidores y bases de datos.
- La excelente herramienta Composer, que simplifica de forma radical la instalación y gestión de las **dependencias** de las aplicaciones PHP, también ha sido creada por varios miembros de la comunidad Symfony.

11. ¿QUE SON LOS BUNDLES EN SYMFONY 2?

Un bundle es un concepto similar al de los plugins en otras aplicaciones, pero todavía mejor. La diferencia clave es que en Symfony2 **todo** es un bundle, incluyendo tanto la funcionalidad básica de la plataforma como el código escrito para tu aplicación.
Los bundles son la parte más importante de Symfony2. Permiten utilizar funcionalidades construidas por terceros o empaquetar tus propias funcionalidades para distribuirlas y reutilizarlas en otros proyectos. Además, facilitan mucho la activación o desactivación de determinadas características dentro de una aplicación.
Un bundle simplemente es un conjunto estructurado de archivos que se encuentran en un directorio y que implementan una sola característica. Puedes crear por ejemplo un BlogBundle, un ForoBundle o un bundle para gestionar usuarios (muchos de ellos ya existen como bundles de software libre). Cada directorio contiene todo lo relacionado con esa característica, incluyendo archivos PHP, plantillas, hojas de estilo, archivos Javascript, tests y cualquier otra cosa necesaria.
Las aplicaciones Symfony se componen de bundles, tal como se define en el método registerBundles() de la clase AppKernel:
Con el método registerBundles(), puedes controlar completamente los bundles que utiliza tu aplicación, incluso aquellos bundles que forman el núcleo del framework.
En reiteradas ocasiones existe la necesidad de implementar librerías de terceros en nuestros proyectos con el objetivo primordial de aprovecharlas, reutilizar código y mejorar nuestros tiempos de entrega; como ya saben en Symfony 2 todo se distribuye en forma de “Bundles” y las librerías de terceros no son la excepción.

Para instalar un bundle:

1.- Registramos los Namespaces en nuestro app/autoload.php:

```php
<?php

use Doctrine\Common\Annotations\AnnotationRegistry;

$loader = require __DIR__.'/../vendor/autoload.php';
require __DIR__.'/../vendor/os/php-excel/PHPExcel/PHPExcel.php';
// intl
if (!function_exists('intl_get_error_code'))
{
    require_once __DIR__.'/../vendor/symfony/symfony/src/Symfony/Component/Locale/Resources/stubs/functions.php';

    $loader->add('', __DIR__.'/../vendor/symfony/symfony/src/Symfony/Component/Locale/Resources/stubs');
}

AnnotationRegistry::registerLoader(array($loader, 'loadClass'));

return $loader;
```

2.- Añadimos el Bundle a nuestro app/AppKernel.php:

```php
<?php

use Symfony\Component\HttpKernel\Kernel;
use Symfony\Component\Config\Loader\LoaderInterface;

class AppKernel extends Kernel
{
    public function registerBundles()
    {
        $bundles = array(
            new Symfony\Bundle\FrameworkBundle\FrameworkBundle(),
            new Symfony\Bundle\SecurityBundle\SecurityBundle(),
            new Symfony\Bundle\TwigBundle\TwigBundle(),
            new Symfony\Bundle\MonologBundle\MonologBundle(),
            new Symfony\Bundle\SwiftmailerBundle\SwiftmailerBundle(),
            new Symfony\Bundle\AsseticBundle\AsseticBundle(),
            new Doctrine\Bundle\DoctrineBundle\DoctrineBundle(),
            new Doctrine\Bundle\FixturesBundle\DoctrineFixturesBundle(),
            new Sensio\Bundle\FrameworkExtraBundle\SensioFrameworkExtraBundle(),
            //new JMS\AopBundle\JMSAopBundle(),
            //new JMS\DiExtraBundle\JMSDiExtraBundle($this),
            //new JMS\SecurityExtraBundle\JMSSecurityExtraBundle(),

            new Sonata\BlockBundle\SonataBlockBundle(),
            new Sonata\CacheBundle\SonataCacheBundle(),
            new Sonata\jQueryBundle\SonatajQueryBundle(),
            new Sonata\AdminBundle\SonataAdminBundle(),
            new Sonata\DoctrineORMAdminBundle\SonataDoctrineORMAdminBundle(),
            new Sonata\EasyExtendsBundle\SonataEasyExtendsBundle(),
            new Sonata\UserBundle\SonataUserBundle('FOSUserBundle'),
            new Application\Sonata\UserBundle\ApplicationSonataUserBundle(),

            new Knp\Bundle\MenuBundle\KnpMenuBundle(),

            new FOS\UserBundle\FOSUserBundle(),
            new FOS\JsRoutingBundle\FOSJsRoutingBundle(),

            new OldSound\RabbitMqBundle\OldSoundRabbitMqBundle(),
            new Knp\Bundle\SnappyBundle\KnpSnappyBundle(),

            //new FR3D\LdapBundle\FR3DLdapBundle(),

            new SunCat\MobileDetectBundle\MobileDetectBundle(),

            new MINSAL\IndicadoresBundle\IndicadoresBundle(),
        );

        if (in_array($this->getEnvironment(), array('dev', 'test'))) {
            $bundles[] = new Symfony\Bundle\WebProfilerBundle\WebProfilerBundle();
            $bundles[] = new Sensio\Bundle\DistributionBundle\SensioDistributionBundle();
            $bundles[] = new Sensio\Bundle\GeneratorBundle\SensioGeneratorBundle();
        }

        return $bundles;
    }

    public function registerContainerConfiguration(LoaderInterface $loader)
    {
        $loader->load(__DIR__.'/config/config_'.$this->getEnvironment().'.yml');
    }
}
```

3.- Configuración adicional al archivo app/config/**config.yml** de la aplicación (la mayoría de los bundles pueden detallar tales configuraciones en su documentación).
Los bundles mas relevantes que usa el eTAB son :

1.- sonata-project / SonataAdminBundle

Genera automáticamente la parte de administración de las aplicaciones. Se integra perfectamente con Doctrine2 y se está mejorando la integración con MongoDB y PHPCR. Su objetivo es convertirse en el "admin generator" oficial de Symfony2.
El funcionamiento del bundle no se basa en archivos de configuración YAML sino en definir una clase de tipo Admin para cada entidad que quieras administrar. Dentro de esa clase, se añaden métodos configureXXXFields() para definir qué propiedades de la entidad se ven en cada página de administración.

2.- sensio / SensioGeneratorBundle

Añade varios comandos útiles para generar diferentes partes de una aplicación Symfony2: bundles, formularios o incluso el esqueleto de la parte de administración o *backend*.
Si tu aplicación utiliza la edición estándar de Symfony2, este bundle ya está incluido y listo para usarse. Ejecuta el comando app/console sin parámetros para ver los comandos incluidos (ver glosario apartado SensioGeneratorBundle)

3.- doctrine / DoctrineBundle

Integra Doctrine2, el ORM más popular de PHP, en las aplicaciones Symfony2. Al principio este bundle era desarrollado por los propios creadores de Symfony2. Sin embargo, para tratar de ser lo más agnóstico posible y para no favorecer a Doctrine frente a Propel, este bundle fue eliminado del núcleo de Symfony2 y ahora se desarrolló de forma independiente.
Además del ORM para trabajar con bases de datos mediante objetos, este bundle también instala el componente DBAL, que permite acceder a cualquier base de datos de forma transparence (MySQL, Oracle, PostgreSQL, SQL Server,...) y ejecutar consultas directamente con SQL.

4.- FriendsOfSymfony / FOSUserBundle

Facilita la gestión de los usuarios en las aplicaciones Symfony2. Se encarga de realizar tareas muy comunes en las aplicaciones que manejan usuarios y que Symfony2 por el momento no soporta. Entre otras, simplifica el registro de usuarios (incluyendo el envío opcional de un email de confirmación) y la opción "olvidé mi contraseña". Compatible con Doctrine, Propel, y MongoDB/CouchDB.

FriendsOfSymfony / FOSJsRoutingBundle

Permite utilizar el sistema de enrutamiento de Symfony directamente desde el código JavaScript de las plantillas. Así puedes generar cualquier ruta de la aplicación con código JavaScript, sin necesidad de pregenerar las rutas con Twig.
Una vez instalado el bundle, sólo debes enlazar los dos siguientes archivos JavaScript en tus plantillas:

```
<script type="text/javascript" src="{{ asset('bundles/fosjsrouting/js/router.js') }}"></script>
<script type="text/javascript" src="{{ path('fos_js_routing_js', {"callback": "fos.Router.setData"}) }}"></script>
```

Ahora ya puedes generar cualquier ruta de la aplicación con el método Router.generate:

```
Routing.generate('homepage');
Routing.generate('noticias_index', { limit: 10 });
Routing.generate('articulo_show', { slug: '...', id: 22 });
```

Para que una ruta esté disponible en JavaScript, debes añadir la opción expose: true en su definición:

```php
<?php

namespace MINSAL\IndicadoresBundle\Controller;

use Symfony\Bundle\FrameworkBundle\Controller\Controller;
use Sensio\Bundle\FrameworkExtraBundle\Configuration\Route;
use Sensio\Bundle\FrameworkExtraBundle\Configuration\ParamConverter;
use Symfony\Component\HttpFoundation\Response;
use Symfony\Component\HttpFoundation\Request;
use MINSAL\IndicadoresBundle\Entity\FichaTecnica;
use MINSAL\IndicadoresBundle\Entity\ClasificacionUso;
use MINSAL\IndicadoresBundle\Entity\User;
use MINSAL\IndicadoresBundle\Entity\GrupoIndicadores;
use MINSAL\IndicadoresBundle\Entity\UsuarioGrupoIndicadores;

class IndicadorController extends Controller
{
    /**
     * @Route("/profile/show", name="fos_user_profile_show")
     */
    public function raiz()
    {
        $this->get('session')->getFlashBag()->add(
                'notice', 'change_password.flash.success'
        );

        return $this->redirect($this->generateUrl('_inicio'));
    }

    /**
     * @Route("/indicador/dimensiones/{id}", name="indicador_dimensiones", options={"expose"=true})
     */
    public function getDimensiones(FichaTecnica $fichaTec)
    {
        $resp = array();
        $em = $this->getDoctrine()->getManager();

        if ($fichaTec)
        {
            $resp['nombre_indicador'] = $fichaTec->getNombre();
            $resp['id_indicador'] = $fichaTec->getId();
            $resp['unidad_medida'] = $fichaTec->getUnidadMedida();
            if($fichaTec->getUpdatedAt()!="")
            $resp["origen_dato_actualizacion"]= date('d/m/Y H:i:s',$fichaTec->getUpdatedAt()->getTimestamp());
            if ($fichaTec->getCamposIndicador() != '')
            {
```

12. ESTRUCTURA DE DIRECTORIOS

Las secciones anteriores explican la filosofía en la que se basa la creación y procesamiento de páginas en Symfony2. También se ha mencionado cómo están estructurados y organizados los proyectos Symfony2. Al final de esta sección, sabrás dónde encontrar y colocar diferentes tipos de archivos y por qué.
Aunque se puede cambiar, por defecto todas las aplicaciones Symfony tienen la misma estructura de directorios sencilla (y recomendada):

- app/: contiene la configuración de la aplicación.
- src/: aquí se encuentra todo el código PHP de la aplicación.
- vendor/: por convención aquí se guardan todas las librerías creadas por terceros.
- web/: este es el directorio web raíz y contiene todos los archivos que se pueden acceder públicamente.

12.1 EL DIRECTORIO WEB

El directorio web raíz es el lugar donde se encuentran todos los archivos públicos y estáticos tales como imágenes, hojas de estilo y archivos JavaScript. También es el lugar donde se definen todos los controladores frontales, como por ejemplo el siguiente:
//web/app.php (Modo Producción) o //web/app_dev.php (Modo developer)

```php
<?php

use Symfony\Component\ClassLoader\ApcClassLoader;
use Symfony\Component\HttpFoundation\Request;

$loader = require_once __DIR__.'/../app/bootstrap.php.cache';

// Use APC for autoloading to improve performance
// Change 'sf2' by the prefix you want in order to prevent key conflict with another application
/*
$loader = new ApcClassLoader('sf2', $loader);
$loader->register(true);
*/

require_once __DIR__.'/../app/AppKernel.php';
//require_once __DIR__.'/../app/AppCache.php';

$kernel = new AppKernel('prod', false);
$kernel->loadClassCache();
//$kernel = new AppCache($kernel);
$request = Request::createFromGlobals();
$response = $kernel->handle($request);
$response->send();
$kernel->terminate($request, $response);
```

El archivo del controlador frontal (app.php en este ejemplo) es el archivo PHP que realmente se ejecuta cuando utilizas una aplicación Symfony2 y su trabajo consiste en *arrancar la aplicación* utilizando una clase del núcleo (AppKernel).
Aunque los controladores frontales son esenciales para servir cada petición, rara vez los tendrás que modificar o incluso pensar en ellos.

12.2 EL DIRECTORIO DE LA APLICACIÓN (APP)

Como vimos en el controlador frontal, la clase AppKernel es el punto de entrada principal de la aplicación y es la responsable de toda la configuración. Como tal, se almacena en el directorio app/. Esta clase debe implementar dos métodos que definen todo lo que Symfony necesita saber acerca de tu aplicación. Ni siquiera tienes que preocuparte de estos métodos durante el arranque — Symfony los rellena por ti con parámetros predeterminados.

- registerBundles(): devuelve un array con todos los bundles necesarios para ejecutar la aplicación.
- registerContainerConfiguration(): carga el archivo de configuración de recursos de la aplicación (consulta la sección *Configurando la aplicación*).

Durante el desarrollo de una aplicación, normalmente el directorio app/ solo los utilizas para modificar la configuración y los archivos de enrutamiento en el directorio app/config/.
Este directorio también contiene el directorio caché de la aplicación (app/cache Nota: LA cache se debe borrar cada vez que hagamos una modificación para que los cambios se reflejen), un directorio de *logs*(app/logs) y un directorio para archivos de recursos globales, tales como plantillas (app/Resources).

12.3 EL DIRECTORIO FUENTE SRC

En pocas palabras, el directorio src/ contiene todo el código real (código PHP, plantillas, archivos de configuración, estilos, etc.) que pertenece a tu aplicación. De hecho, al programar una aplicación Symfony, la mayor parte de tu trabajo se llevará a cabo dentro de uno o más bundles creados en este directorio.
Bundles

1. Application: Contiene unos archivos que hacen overrider a una parte especifica del algún vendor.
2. MINSAL: Nuestro proyecto en si.

El directorio IndicadoresBundle (Nuestro bundle principal) tiene la siguiente estructura:

1. Admin
2. Command
3. Consumer
4. Controller
5. DataFixture
6. DependencyInjection
7. Entity
8. Excel
9. Filter
10. Listener
11. Resources
12. Test
13. Util
14. Validadtor

12.3.1 ADMIN

Esta carpeta contiene todos los archivos de configuración para crear los formularios y los Grid de cada uno de los módulos que conforman el eTAB.

Contiene los siguientes métodos:

protected function configureFormFields(FormMapper $formMapper): Este método configure los campos que se incluyen en el formulario.
->with Crea una pestaña nueva (tab panel)
->add Crea el control que se añade a la pestaña este recibe 3 parámetros

1. Nombre: Este debe ser el mismo nombre del campo con el que se está asociando en la entity.
2. Tipo de campo: null campo por defecto definido por la entity, se pueden utilizar los tipos como los conocemos en HTML

3. Array: Colección con más opciones como etiquetas(Label), clases otros códigos HTML

```
protected function configureFormFields(FormMapper $formMapper)
{
    $formMapper
            ->with($this->getTranslator()->trans('_datos_generales_'))
            ->add('nombre', null, array('label' => $this->getTranslator()->trans('nombre_indicador')))
            ->add('tema', null, array('label' => $this->getTranslator()->trans('_interpretacion_')))
            ->add('concepto', null, array('label' => $this->getTranslator()->trans('concepto')))
            ->add('unidadMedida', null, array('label' => $this->getTranslator()->trans('unidad_medida')))
            ->add('esAcumulado', null, array('label' => $this->getTranslator()->trans('es_acumulado')))
            ->add('esPublico', null, array('label' => $this->getTranslator()->trans('es_publico')))
            ->add('variables', null, array('label' => $this->getTranslator()->trans('variables'), 'expanded' => true))
```

protected function configureShowFields(ShowMapper $showMapper): Este método se configura igual que el anterior. Crea la vista show.

protected function configureDatagridFilters(DatagridMapper $datagridMapper): Este método crea los flitros en se ultilizaran en el grid tambien tiene un metodo ->add con los mismos parametros

```
protected function configureDatagridFilters(DatagridMapper $datagridMapper)
{
    $datagridMapper
            ->add('nombre', null, array('label' => $this->getTranslator()->trans('nombre_indicador')))
            ->add('clasificacionTecnica', null, array('label' => $this->getTranslator()->trans('clasificacion_tecnica')))
            ->add('clasificacionPrivacidad', null, array('label' => $this->getTranslator()->trans('_nivel_de_usuario_')))
    ;
}
```

protected function configureListFields(ListMapper $listMapper): Este método devuelve el grid

```
protected function configureListFields(ListMapper $listMapper)
{
    $listMapper
            ->addIdentifier('nombre', null, array('label' => $this->getTranslator()->trans('nombre_indicador')))
            ->add('tema', null, array('label' => $this->getTranslator()->trans('_interpretacion_')))
            ->add('meta', null, array('label' => $this->getTranslator()->trans('meta')))
            ->add('camposIndicador', null, array('label' => $this->getTranslator()->trans('campos_indicador')))
            ->add('clasificacionTecnica', null, array('label' => $this->getTranslator()->trans('clasificacion_tecnica')))
            ->add('clasificacionPrivacidad', null, array('label' => $this->getTranslator()->trans('_nivel_de_usuario_')))
```

public function getTemplate($name): Aca se configura los templetes para cada caso de uso.

```
public function getTemplate($name)
{
    switch ($name) {
        case 'edit':
            return 'IndicadoresBundle:CRUD:ficha_tecnica-edit.html.twig';
            break;
        case 'show':
            return 'IndicadoresBundle:FichaTecnicaAdmin:show.html.twig';
            break;
        default:
            return parent::getTemplate($name);
            break;
    }
}
```

public function validate(ErrorElement $errorElement, $object): Validación de los formularios. Esta validación también se puede hacer en la entity o en el controlador.

12.3.2 CONTROLLER

Un controlador es una función PHP creada por ti y que se encarga de obtener la información de la petición HTTP y de generar y devolver la respuesta HTTP (en forma de objeto de tipo Response de Symfony2). La respuesta puede ser una página HTML, un documento XML, un Array, JSON serializado, una imagen, una redirección a otra página, un error de tipo 404 o cualquier otra cosa que se te ocurra. El controlador contiene toda la lógica que tu aplicación necesita para generar el contenido de la página.

12.3.3 ENTITY

Esta carpeta contiene todos los modelos y objetos ORM divididos en 2 el primero nombreEntity corresponde al modelo que se forma por el nombre de los campos tipo private de la base de datos y que se pueden acceder a ellos por los métodos setter y getter public para cada campo; Se utilizan parámetros Annotation para configurar la base de datos.

El segundo nombreEntityRepository normalmente se usa para crear consultas que la propia entity no nos proporciona. Tiene configurado métodos por default como son finAll(), finById(), finByCampo() (Campo = nombre de algún campo de la entity por ejemplo nombre, descripción, etc)

Nota: Un repository puede estar físicamente o no en la carpeta entity, esto depende de que si necesitemos añadir alguna función extra que no se encuentre en el repository por default.

Ejemplo: de configuración de variables con parámetros Annotation para afectar a una tabla en la base de datos.

```php
<?php
namespace MINSAL\IndicadoresBundle\Entity;
use Doctrine\ORM\Mapping as ORM;
use MINSAL\IndicadoresBundle\Validator as CustomAssert;
/**
 * MINSAL\IndicadoresBundle\Entity\Alerta
 *
 * @ORM\Table(name="alerta")
 * @ORM\Entity
 */
class Alerta
{
    /**
     * @var integer $id
     *
     * @ORM\Column(name="id", type="integer", nullable=false)
     * @ORM\Id
     * @ORM\GeneratedValue(strategy="AUTO")
     */
    private $id;

    /**
     * @var string $codigo
     *
     * @ORM\Column(name="codigo", type="string", length=50, nullable=false)
     * @CustomAssert\ValidHTMLcolor(message="ValidHTMLcolor.Message")
     */
    private $codigo;

    /**
     * @var string $color
     *
     * @ORM\Column(name="color", type="string", length=50, nullable=false)
     * @CustomAssert\OnlyAlphanumeric(message="OnlyAlphanumeric.Message")
     */
```

Ejemplo: Metodos setter y getter para las variables, En algunos caso al devolver una cadena es necesario hacerlo por el método __toString() para que no tenga problemas en retornar el valor como tal.

```
47
48      /**
49       * Set color
50       *
51       * @param  string $color
52       * @return Alerta
53       */
54      public function setColor($color)
55      {
56          $this->color = $color;
57
58          return $this;
59      }
60
61      /**
62       * Get color
63       *
64       * @return string
65       */
66      public function getColor()
67      {
68          return $this->color;
69      }
70
71      public function __toString()
72      {
73          return $this->color ? :'';
74      }
75
76      /**
77       * Set codigo
78       *
79       * @param  string $codigo
80       * @return Alerta
81       */
82      public function setCodigo($codigo)
```

Ejemplo: de un Repository

```
1   <?php
2
3   namespace MINSAL\IndicadoresBundle\Entity;
4
5   use Doctrine\ORM\EntityRepository;
6
7   /**
8    * BoletinRepository
9    *
10  * This class was generated by the Doctrine ORM. Add your own custom
11  * repository methods below.
12  */
13  class BoletinRepository extends EntityRepository
14  {
15      public function getUsuarioGrupo($grupo)
16      {
17          $stmt = $this->getEntityManager()
18          ->getConnection()
19          ->prepare("SELECT * FROM fos_user_user u left join fos_user_user_group g on g.user_id=u.id where g.group_id='".$grupo->getId().
    "'");
20          $stmt->execute();
21          return $stmt->fetchAll();
22      }
23      public function getRuta($sala,$token)
24      {
25          $stmt = $this->getEntityManager()
26          ->getConnection()
27          ->prepare("SELECT * FROM boletin b left join grupo_indicadores s on s.id=b.sala where b.token='$token' and ".((gettype($sala) ==
    'integer') ? "s.id=".$sala :  "s.nombre='".$sala."'"));
28          $stmt->execute();
29          $result= $stmt->fetchAll();
30
31          foreach($result as $res)
32          {
33              $tiempo=$this->duracion(\DateTime::createFromFormat('Y-m-d H:i:s',$res['actualizado']));
34              if(stripos($tiempo,'dia'))
```

12.3.4 RESOURCES

Esta carpeta está dividida en 4 subdirectorios.
Config: Contiene las configuraciones particulares del bunlde (IndicadoresBunlde).

Public: Contiene todos los archivos JavaScript y hojas de estilo CSS además se encuentran todos las imágenes.
Translations: Contiene los archivos de traducción.
Views: Contiene todas las vistas del proyecto en plantillas TWIG ¿Por qué Twig? Porque las plantillas en Twig son muy fáciles de hacer y resultan muy intuitivas en el caso de que contemos con maquetadores o Diseñadores Frontend, además de todo ello su sintaxis corta y concisa.

La plantilla principal para todos los catalogos es standard_layout.html.twig
La plantilla principal para el tablero público es standard_layout_public.html.twig
La plantilla para el tablero es standard_layout_tablero.html.twig

12.3.5 UTIL

Contiene el script que inicia las colas para la carga masiva de datos con RabbitMQ.

12.3.6 VALIDATOR

La validación es una tarea muy común en aplicaciones web. Los datos introducidos en los formularios deben ser validados. Los datos también deben ser validados antes de que se escribe en una base de datos o se pasa a un servicio web.

Se crean archivos personalizados que extienden de la clase base, Constraint. Como ejemplo se va a analizar un validador simple que comprueba si una cadena contiene sólo caracteres alfanuméricos. Para esto se usan los archivos AlphanumericPlusValidator.php(Contiene la validación) y AlphanumericPlus.php(Contiene el mensaje, en caso de que la validación no se cumpla).

AlphanumericPlusValidator.php

```php
<?php
namespace MINSAL\IndicadoresBundle\Validator;

use Symfony\Component\Validator\Constraint;
use Symfony\Component\Validator\ConstraintValidator;

class AlphanumericPlusValidator extends ConstraintValidator
{
    public function validate($value, Constraint $constraint)
    {
        if(!empty($value)){
            if (!preg_match('/^[a-zA-Za0-9 áéíóúüÁÉÍÓÚÜ\'ñÑ.,\r\n_\-%]+$/', $value, $matches)) {
                $this->context->addViolation($constraint->message, array('%string%' => $value));
            }
        }
    }
}
?>
```

AlphanumericPlus.php

```php
<?php
namespace MINSAL\IndicadoresBundle\Validator;

use Symfony\Component\Validator\Constraint;

/**
 * @Annotation
 */
    class AlphanumericPlus extends Constraint
    {
        public $message = 'The string "%string%" contains an illegal character: it can only contain letters, numbers or (. , \' ñ) or accented vowels or diaeresis.';
    }
?>
```

El @Annotation es necesario para que la nueva restricción esté disponible para su uso en las clases a través de anotaciones

13. CONFIGURANDO EL ETAB

En este capítulo vamos a conocer la estructura de la configuración de los módulos que componen al eTAB.

13.1 LOS ARCHIVOS DE CONFIGURACIÓN

Config.yml: Este archivo contiene la configuración global del proyecto (hay un archivo de configuración para el bundle). En este archivo lo más importante que debemos tener en cuenta es la configuración de sonata_admin y fos_user ya que estos muestran todo el modelo del backend y control de permisos y usuario.

```yaml
sonata_admin:
    security:
        handler: sonata.admin.security.handler.role
    title: ' '
    title_logo: bundles/indicadores/images/logo_salud.png
    templates:
        # default global templates
        layout:  IndicadoresBundle::standard_layout.html.twig
        ajax:    SonataAdminBundle::ajax_layout.html.twig

        # default actions templates, should extend a global templates
        list:    IndicadoresBundle:CRUD:list.html.twig
        show:    IndicadoresBundle:CRUD:show.html.twig
        edit:    IndicadoresBundle:CRUD:edit.html.twig
        # se personalizó la plantilla de borrar
        delete:  IndicadoresBundle:CRUD:delete.html.twig
        list_block: IndicadoresBundle:Block:block_admin_list.html.twig


fos_user:
    db_driver:        orm # can be orm or odm
    firewall_name:    main
    user_class:       MINSAL\IndicadoresBundle\Entity\User
    group:
        group_class:  Application\Sonata\UserBundle\Entity\Group

    profile:  # Authentication Form
        form:
            type:               fos_user_profile
            handler:            fos_user.profile.form.handler.default
            name:               fos_user_profile_form
            validation_groups:  [Authentication] # Please note : this is not the default value
```

Lo otro a tener en cuenta es la configuración del envío de correos para lo cual tenemos que modificar estas líneas.

```
mobile_detect:
    redirect:
        mobile: ~
        tablet: ~
    switch_device_view: ~

swiftmailer:
    transport:  gmail
    #encryption: ssl
    #auth_mode:  login
    #host:      smtp.gmail.com
    username:  boletin.dev.test@gmail.com
    password:  Boletin@test
    #spool:     { type: memory }
    #disable_delivery:  true
    #schiapassm2015@gmail.com
    #bit@2013 (B mayuscula o minuscula)
```

Parameter.yml: Este archivo contiene la configuración de conexión a la base de datos este se configura automáticamente en el proceso de instalación. Pero si por algún motivo queremos cambiar la conexión tenemos que modificar sus parámetros manualmente.

```
# This file is auto-generated during the composer install
parameters:
    database_driver: pdo_pgsql
    database_host: localhost
    database_port: null
    database_name: indicadores_6
    database_user: admin
    database_password: rodriguez
    mailer_transport: smtp
    mailer_host: localhost
    mailer_user: null
    mailer_password: null
    locale: es_MX
    secret: 295125e6c66ab2a1038b62ad3c910733510
    database_path: null
    archivo_vitacora: '%kernel.logs_dir%/minsal.log'
    carpeta_siig_mondrian: '%kernel.root_dir%/mondrian/'
    conexion_bd_pentaho: Minsal
    listado_metadata: datasources.siig
```

Routing.yml: Se utiliza para generar todas las rutas que se utilizaran en el controlador o las vistas dependiendo su uso. Todas las rutas deben de tener un controlador asociado.

Si bien este es el archivo para las rutas pero no se encuentran todas las que se utilizan para el eTAB dado que se está utilizando el bundle **FOSJsRoutingBundle** que hace que podamos insertar código Annotation sobre algún método del controlador y generar la ruta desde su instancia.

Ejemplo de la implementación del bundle **FOSJsRoutingBundle**

```
{
    /**
    * @Route("/publico/boletin/{sala}/{token}", name="boletin_publico", options={"expose"=true})
    */
    public function tokenAction($sala,$token)
    {
        $em = $this->getDoctrine()->getManager();
```

El archivo ruting

```
_inicio:
    pattern:  /
    defaults: { _controller: IndicadoresBundle:Tablero:tablero }

_inicioPublico:
    pattern:  /
    defaults: { _controller: IndicadoresBundle:Tablero:tableroPublico }

group_show:
    pattern:  /migroup/{token}/{valor}
    defaults: { _controller: IndicadoresBundle:Indicador:group_reload }
    requirements:
        _method:  GET

indicadores:
    resource: "@IndicadoresBundle/Controller/"
    type:     annotation
    prefix:   /

# Internal routing configuration to handle ESI
#_internal:
#    resource: "@FrameworkBundle/Resources/config/routing/internal.xml"
#    prefix:   /_internal
admin:
    resource: '@SonataAdminBundle/Resources/config/routing/sonata_admin.xml'
    prefix: /admin

_sonata_admin:
    resource: .
    type: sonata_admin
    prefix: /admin

sonata_page_cache:
    resource: '@SonataCacheBundle/Resources/config/routing/cache.xml'
    prefix: /

fos_user_security:
    resource: "@FOSUserBundle/Resources/config/routing/security.xml"
```

Security.yml: El componente de seguridad se configura mediante el archivo de configuración principal de la aplicación. Para proteger tu aplicación, solo es necesario configurar unas pocas opciones de su archivo de configuración.

Los apartados más relevantes son:

firewalls: Acá se configura la parte de la ruta que se va a pedir autenticación. En el archivo de configuración encontramos que admin: pattern; /admin(.*) esto significa que para todos las rutas que contengan el admin se le pida las credenciales.

```
    firewalls:
        # -> custom firewall for the admin area of the URL
        admin:
            switch_user:        true
            context:            user
            pattern:            /admin(.*)
            #fr3d_ldap:           ~
            form_login:
                login_path:     /admin/login
                use_forward:    false
                check_path:     /admin/login_check
                default_target_path: /admin/dashboard
                use_referer:    true
            logout:
                path:           /admin/logout
                target:         /admin/login

            anonymous:    true
        # -> end custom configuration

        # default login area for standard users
        main:
            switch_user:        true
            context:            user
            pattern:            .*
            #fr3d_ldap:           ~
            form_login:
                login_path:     /admin/login
                use_forward:    false
                check_path:     /admin/login_check
                failure_path:   null
                default_target_path: /admin/dashboard
                use_referer:    true
            logout:             true
            anonymous:          true
```

access_control: En este apartado ponemos las excepciones para el caso de las rutas que no necesitamos que se nos pida autorización o acceso ya que necesitamos acceder a ellas y por alguna razón no estar logueados como es el caso del tablero público.

```
    access_control:
        # URL of FOSUserBundle which need to be available to anonymous users
        - { path: ^/_wdt, role: IS_AUTHENTICATED_ANONYMOUSLY }
        - { path: ^/_profiler, role: IS_AUTHENTICATED_ANONYMOUSLY }
        - { path: ^/login$, role: IS_AUTHENTICATED_ANONYMOUSLY }
        - { path: ^/js/, role: IS_AUTHENTICATED_ANONYMOUSLY }

        # -> custom access control for the admin area of the URL
        - { path: ^/admin/login$, role: IS_AUTHENTICATED_ANONYMOUSLY }
        - { path: ^/admin/logout$, role: IS_AUTHENTICATED_ANONYMOUSLY }
        - { path: ^/admin/login-check$, role: IS_AUTHENTICATED_ANONYMOUSLY }
        - { path: ^/publico, role: IS_AUTHENTICATED_ANONYMOUSLY }
        - { path: ^/indicador/datos/public/, role: IS_AUTHENTICATED_ANONYMOUSLY }
        - { path: ^/indicador/dimensiones/public/, role: IS_AUTHENTICATED_ANONYMOUSLY }
        - { path: ^/indicador/filtrar/datos/public, role: IS_AUTHENTICATED_ANONYMOUSLY }
        - { path: ^/indicador/_locale$/change, role: IS_AUTHENTICATED_ANONYMOUSLY }
        # -> end
```

Services.xml: Este archivo contiene los datos para crear el menú.

```
<container xmlns="http://symfony.com/schema/dic/services"
           xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
           xsi:schemaLocation="http://symfony.com/schema/dic/services/services-1.0.xsd">
    <parameters>
        <parameter key="admin_max_per_page_number">10</parameter>
    </parameters>
    <services>

        <service id="sonata.admin.fuente_dato" class="MINSAL\IndicadoresBundle\Admin\FuenteDatoAdmin">
            <tag name="sonata.admin"
                 manager_type="orm"
                 group="catalogos"
                 label="fuente_datos-eTAB"/>
            <argument />
            <argument>MINSAL\IndicadoresBundle\Entity\FuenteDato</argument>
            <argument>SonataAdminBundle:CRUD</argument>
            <call method="setMaxPerPage"><argument>%admin_max_per_page_number%</argument></call>
        </service>

        <service id="sonata.admin.responsable_dato" class="MINSAL\IndicadoresBundle\Admin\ResponsableDatoAdmin">
            <tag name="sonata.admin"
                 manager_type="orm"
                 group="catalogos"
                 label="responsable_datos-eTAB"/>
            <argument />
            <argument>MINSAL\IndicadoresBundle\Entity\ResponsableDato</argument>
            <argument>SonataAdminBundle:CRUD</argument>
            <call method="setMaxPerPage"><argument>%admin_max_per_page_number%</argument></call>
        </service>

        <service id="sonata.admin.responsable_indicador" class="MINSAL\IndicadoresBundle\Admin\ResponsableIndicadorAdmin">
            <tag name="sonata.admin"
                 manager_type="orm"
                 group="catalogos"
                 label="responsable_indicador-eTAB"/>
            <argument />
            <argument>MINSAL\IndicadoresBundle\Entity\ResponsableIndicador</argument>
```

13.2 LA TRADUCCIÓN.

Los archivos de traducción que se utilizan para el cambio de lenguaje se encuentran en la ruta src\MINSAL\IndicadoresBundle\Resources\traslation

Los archivos con el prefijo messages es la que se utiliza para la traducción de la mayoría de los mensajes en el proyecto.

Para el caso de queramos traducir algo en JavaScript necesitamos agregarlo en la el archivo messages_js.html que lo encontramos en la ruta src\MINSAL\IndicadoresBundle\Resources\views y agregarlo también en su respectivos messages en el apartado javascript:

```
javascript:
  elija_tipo_dato: 'Elija el tipo de dato'
  elija_significado_dato: 'Elija el significado del dato'
  configure_campos: 'Configure los campos: se muestra solo una parte de los datos'
  nombre_campo: 'Nombre del campo'
  tipo: 'Tipo'
  significado: 'Significado'
  datos_muestra: 'Datos de muestra'
  dimension: 'Dimensión'
  tipo_grafico: 'Tipo de gráfico'
  ascendente: 'Ascendente'
```

13.3 LAS VISTAS

Las vistas las encontramos en la carpeta src\MINSAL\IndicadoresBundle\Resources\views la plantilla principal para cada caso es las que tiene el prefijo estándar_layout en este directorio encontramos una serie de subdirectorios pero el más importante o el que más impacto tiene es el que tiene por nombre CRUD en este directorio se encuentra las plantillas que reciben los datos para crear los formularios, listados, filtros y ver.

14 EL TABLERO DE CONTROL

La parte más importante del eTAB es el tablero ya que todas las configuraciones no tienen caso si no podemos visualizar los resultados en las gráficas.

Pero como podemos modificar una gráfica o cambiar el comportamiento de otra. En este capítulo vamos adentrarnos a la programación del tablero desde que se construye hasta como interactuamos con él.

14.1 LOS ARCHIVOS DEL DASHBOARD

Estos archivos interactúan entre sí para que el tablero pueda mostrar las gráficas en la forma como se las presenta al usuario.

FichaTecnicaAdminController.php (src\MINSAL\IndicadoresBundle\Controller): Genera la vista

IndicadorController.php (src\MINSAL\IndicadoresBundle\Controller): Este archivo es el responsable de generar todos los datos que se muestran en las gráficas, con su respectivo filtro u opción que el usuario que este interactuando con el tablero elija.

Tablero.html.twig (src\MINSAL\IndicadoresBundle\Resources\views\FichaTecnicaAdmin): Este archivo se encarga de mostrar al usuario la vista donde se alojaran las gráficas.

Tablero.js (src\MINSAL\IndicadoresBundle\Resources\public\js\FichaTecnicaAdmin): Una vez que tengamos donde poner cada área de gráficos, este archivo se encarga de darle forma a las áreas de gráficos así como las acciones para cada botón sala e indicadores.

Común.js (src\MINSAL\IndicadoresBundle\Resources\public\js\FichaTecnicaAdmin): Teniendo las áreas de gráficos este se encarga de ponerle los menús y las acciones para cada uno. Con su respectiva gráfica.

Para aquellos archivos que tienen el sufijo _public son copias de los archivos sin sufijo esto quiere decir que por cada modificación que realicemos a estos archivos también las tenemos que considerar en replicarlos en los que sí tienen el sufijo _public.

Grafico_(tipo).js (src\MINSAL\IndicadoresBundle\Resources\public\js\FichaTecnicaAdmin): Donde tipo puede tomar el valor de columnas, líneas o pastel y su obligación es dibujar el gráfico con la ayuda de la librería d3.js y además agrega el evento clic y touch para desagregar un nivel en el gráfico.

14.2 LA PROGRAMACIÓN

14.2.1 EL CONTROLADOR QUE INICIA EL TABLERO

Para acceder al tablero nos vamos a la siguiente ruta:

admin/minsal/indicadores/fichatecnica/tablero Que está asociado al controlador **FichaTecnicaAdminController.php** y que ejecuta el método **public function tableroAction()** este método se encarga de obtener toda la información del usuario necesario para la generación de los menús y opciones a mostrar en el tablero, como son las clasificaciones por rol y usuario, indicadores por rol y por usuario y las salas que ha creado o en las que participa.

```php
<?php

namespace MINSAL\IndicadoresBundle\Controller;

use Sonata\AdminBundle\Controller\CRUDController as Controller;
use Symfony\Component\HttpFoundation\Response;
use Symfony\Component\HttpFoundation\RedirectResponse;
//use Symfony\Component\Console\Input\ArrayInput;

class FichaTecnicaAdminController extends Controller
{
    public function editAction($id = null)
    {
        $repo = $this->getDoctrine()->getManager()->getRepository('IndicadoresBundle:FichaTecnica');
        $this->admin->setRepository($repo);

        return parent::editAction($id);
    }

    public function createAction()
    {
        $repo = $this->getDoctrine()->getManager()->getRepository('IndicadoresBundle:FichaTecnica');
        $this->admin->setRepository($repo);

        return parent::createAction();
    }

    public function tableroAction()
    {
        $user = $this->container->get('security.context')->getToken()->getUser();
        $em = $this->getDoctrine()->getManager();
        $clasificacionUso = $em->getRepository("IndicadoresBundle:ClasificacionUso")->findAll();

        //Luego agregar un método para obtener la clasificacion de uso por defecto del usuario
        $usuario = $this->getUser();
        if ($usuario->getClasificacionUso()) {
            $clasificacionUsoPorDefecto = $usuario->getClasificacionUso();
        }
        else
        {
            $clasificacionUsoPorDefecto = $clasificacionUso[0];
        }
        $categorias = $em->getRepository("IndicadoresBundle:ClasificacionTecnica")->findBy(array('clasificacionUso' => $clasificacionUsoPorDefecto));
```

Una vez que obtiene todo lo necesario le indica al sistema que cargue toda la informacion en la vista **tablero.html.twig** que se encuentra en la carpeta FichaTecnicaAdmin en Resources/views, los datos se pasan como un array associativo que definen el contenido de los menus indicadores y salas

```
155
156         return $this->render('IndicadoresBundle:FichaTecnicaAdmin:tablero.html.twig', array(
157                     'categorias' => $categorias_indicador,
158                     'clasificacionUso' => $clasificacionUso,
159                     'salas' => $salas,
160                     'salasXusuario' => $salasXusuario,
161                     'salasXgrupo' => $salasXgrupo,
162                     'indicadores_no_clasificados' => $indicadores_no_clasificados
163         ));
164     }
```

14.2.2 LA VISTA

La vista **tablero.html.twig** se encuentra en la ruta:

src\MINSAL\IndicadoresBundle\Resources\views\FichaTecnicaAdmin y extiende de la plantilla {% extends 'IndicadoresBundle::standard_layout_tablero.html.twig' %} (plantilla principal tablero).

Anexa librerías js en el block {% block javascripts %}.

También extiende del menú para el tablero:

{% include 'IndicadoresBundle:FichaTecnicaAdmin:menu_tablero.html.twig' %} el cual se encarga de dibujar los controles siguientes:

En **tablero.html.twig** también se encuentra las estructuras de las ventanas modales para cada uno de estos controles.

14.2.3 EL MANEJADOR DE EVENTOS PARA LOS CONTROLES DEL TABLERO

En la plantilla se carga el archivo **tablero.js** y otras librerías que se explican más adelante.

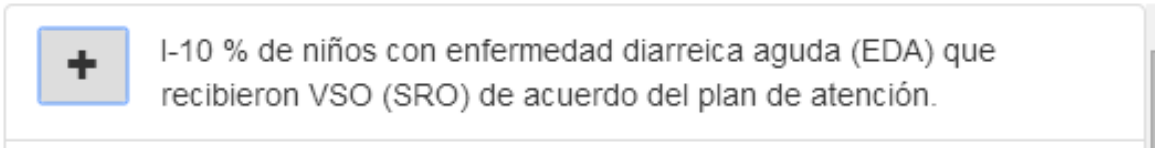
```
{% block javascripts %}
    {{ parent() }}
<script src="{{ asset('bundles/indicadores/js/d3.min.js') }}" type="text/javascript"></script>
<script src="{{ asset('bundles/indicadores/js/d3pie.js') }}" type="text/javascript"></script>
<script src="{{ asset('bundles/indicadores/js/affix.min.js') }}" type="text/javascript"></script>

<script src="{{ asset('bundles/indicadores/js/FichaTecnicaAdmin/grafico_pastel.js') }}" type="text/javascript"></script>
<script src="{{ asset('bundles/indicadores/js/FichaTecnicaAdmin/grafico_columnas.js') }}" type="text/javascript"></script>
<script src="{{ asset('bundles/indicadores/js/FichaTecnicaAdmin/grafico_lineas.js') }}" type="text/javascript"></script>
<script src="{{ asset('bundles/indicadores/js/FichaTecnicaAdmin/tablero.js') }}" type="text/javascript"></script>
<script src="{{ asset('bundles/indicadores/js/FichaTecnicaAdmin/comun.js') }}" type="text/javascript"></script>
```

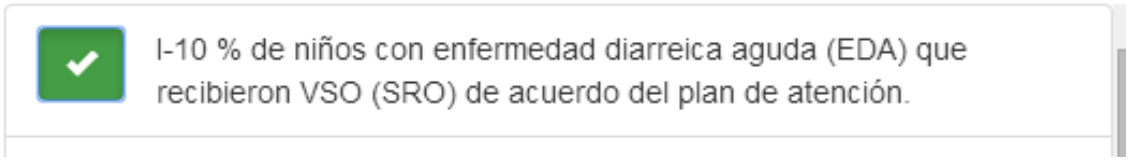
El tablero.js es el encargado de los llamados Ajax para el menú (indicadores y sala) este archivo se encuentra ubicado en:

src\MINSAL\IndicadoresBundle\Resources\public\js\FichaTecnicaAdmin y contiene las siguientes funciones.

$('.indicador').on('click',function(e) Está asociado al clic de los botones con el icono de agregar para cada indicador. Primero obtiene el id del indicador y se lo pasa como parámetros a la función **cargar_indicador(mid);**



function cargar_indicador(mid) Esta función tiene una condición si el indicador ya está agregado obtiene el id del área de gráfico y lo limpia de la zona de gráficos **limpiarZona2(id);** (función del archivo común) si no agrega el botón guardar sala al menú y llama al método **sala_agregar_fila();** , cambia el botón a activo y dibuja el indicador **dibujarIndicador($("#"+mid).attr('data-id'));**



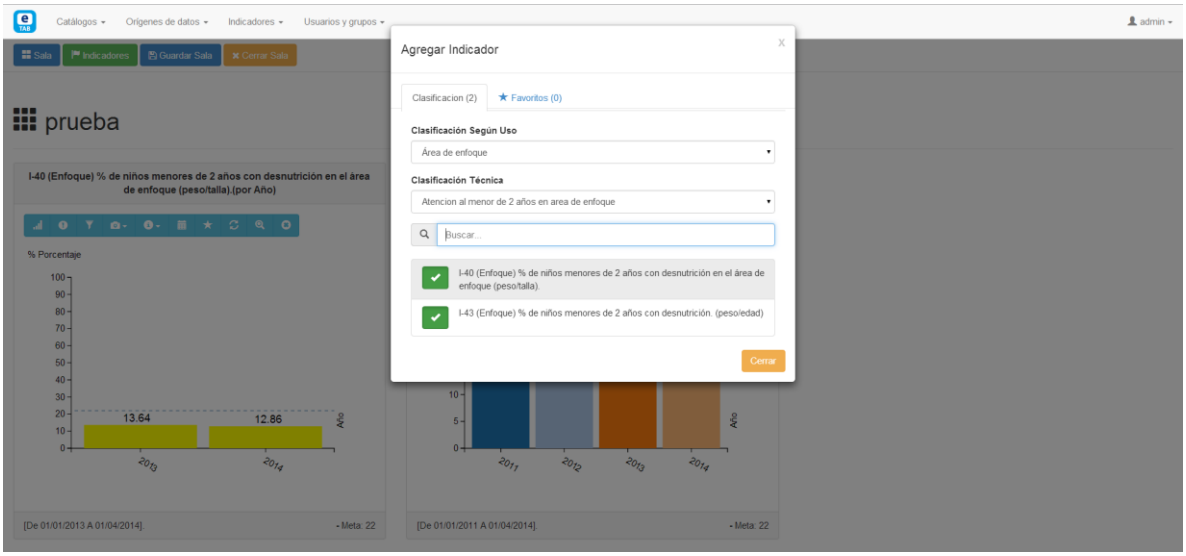
function limpiarZona2(id) : Encargado de remover un indicador del área de gráficos, además reacomoda los gráficos para que no queden rows vacios.

function sala_agregar_fila() : Agrega el área de grafico donde se dibujara la gráfica y su menú.

function dibujarIndicador(id_indicador) : Esta función únicamente llama a otra función del archivo común.js **recuperarDimensiones(id_indicador, null);** con un parámetro que es el id del indicador a cargar. Más a delante se explica la funcionalidad de esta función**.**

$("#mimodal").on('click',function(e) : llama a la función **marcar_agregados();**

function marcar_agregados() : Marca las casillas de los indicadores que aparezcan cargados en el área de gráfico.

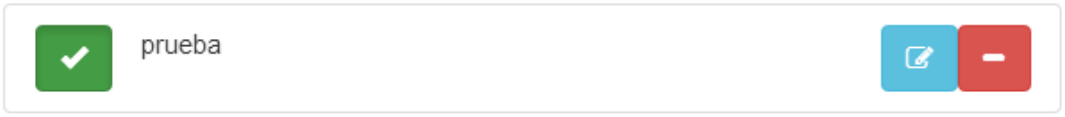


$("#_cerrar_sala_").on('click',function(e) : Cierra una sala y desmarca todos los indicadores de la lista de selección.

$('.salas-id').on('click',function(e) : Si una sala ya está agregada al área de gráficos; permite quitarlo cerrando todos los indicadores que le pertenezcan y cambia el icono para poder agregarlo nuevamente.

Si no está agregado; permite agregar todos sus indicadores en el área de gráfico.

$('#elimina_sala').on('click',function(e) : evento para el botón eliminar sala



Elimina una sala del usuario con sus respectivos indicadores. Llama al siguiente método del controlador **indicadorcontroller.php**

$.getJSON(Routing.generate('sala_eliminar'), {datos: JSON.stringify(datos_sala)},

Si se obtiene una respuesta verdadera se quitan la sala de la lista de selección. Y se muestra un mensaje al usuario.

Si no se muestra un mensaje de error.

```
/**
 * @Route("/sala/eliminar", name="sala_eliminar", options={"expose"=true})
 */
public function eliminaSala()
{
    $lasala=new GrupoIndicadores();
    $em = $this->getDoctrine()->getManager();
    $req = $this->getRequest();
    $resp = array();
    $em->getConnection()->beginTransaction();
    $sala = json_decode($req->get('datos'));
    try {
        if ($sala->id != '')
        {
            $grupoIndicadores = $em->find('IndicadoresBundle:GrupoIndicadores', $sala->id);
            //Borrar los indicadores antiguos de la sala
            foreach ($grupoIndicadores->getIndicadores() as $ind)
                $em->remove($ind);

            foreach ($grupoIndicadores->getUsuarios() as $ind)
                $em->remove($ind);

            $lasala=$em->find('IndicadoresBundle:GrupoIndicadores', $sala->id);
            $em->remove($lasala);

            $em->flush();

            $resp['estado'] = 'ok';
            $em->getConnection()->commit();
        }
        else
        {
            $resp['estado'] = 'error';
        }
    }
    catch (Exception $e)
    {
        $em->getConnection()->rollback();
        $em->close();
        $resp['estado'] = 'error';
        throw $e;
    }
    return new Response(json_encode($resp));
}
```

$('#guardar_sala').on('click',function(e) : Guarda o actualiza los datos de una sala; Incluye los filtros, las gráficas y las opciones que el usuario tenga seleccionado en cada una.

Llama la función con Ajax del controlador **indicadorController.php**

$.getJSON(Routing.generate('sala_guardar'), {datos: JSON.stringify(datos_sala)},

```
/**
 * @Route("/sala/guardar", name="sala_guardar", options={"expose"=true})
 */
public function guardarSala()
{
    $em = $this->getDoctrine()->getManager();
    $req = $this->getRequest();
    $resp = array();

    $sala = json_decode($req->get('datos'));
    $em->getConnection()->beginTransaction();
```

14.2.4 EL MENÚ DE CADA GRÁFICA

Este archivo está asociado a todos los eventos de cada gráfica, en la sección pasada se hace mención del llamado a esta función:

function recuperarDimensiones(id_indicador, datos): Se encarga de cargar los datos para el id del indicador que se le pase como primer parámetro, el segundo parámetro son datos de filtrado.

Primero llama a la función **limpiarZona(zona).** Le pasa como parámetro el id del área de grafico seleccionada.

Después hace un llamado Ajax al controlador **indicadorController.php**

$.getJSON(Routing.generate('indicador_dimensiones', {id: id_indicador}),

```
29
30      /**
31       * @Route("/indicador/dimensiones/{id}", name="indicador_dimensiones", options={"expose"=true})
32       */
33      public function getDimensiones(FichaTecnica $fichaTec)
34      {
35          $resp = array();
36          $em = $this->getDoctrine()->getManager();
37
38          if ($fichaTec)
39          {
40              $resp['nombre_indicador'] = $fichaTec->getNombre();
41              $resp['id_indicador'] = $fichaTec->getId();
42              $resp['unidad_medida'] = $fichaTec->getUnidadMedida();
43              if($fichaTec->getUpdatedAt()!="")
44              $resp["origen_dato_actualizacion"]= date('d/m/Y H:i:s',$fichaTec->getUpdatedAt()->getTimestamp());
45              if ($fichaTec->getCamposIndicador() != '')
46              {
47                  $campos = explode(',', str_replace(array("'", ' '), array('', ''), $fichaTec->getCamposIndicador()));
48              }
49              else
50              {
51                  $campos = array();
52              }
53              $dimensiones = array();
54              foreach ($campos as $campo)
55              {
56                  $significado = $em->getRepository('IndicadoresBundle:SignificadoCampo')->findOneByCodigo($campo);
57                  if (count($significado->getTiposGraficosArray()) > 0)
58                  {
59                      $dimensiones[$significado->getCodigo()]['descripcion'] = ucfirst(preg_replace('/^Identificador /i', '', $significado->getDescripcion()));
60                      $dimensiones[$significado->getCodigo()]['escala'] = $significado->getEscala();
61                      $dimensiones[$significado->getCodigo()]['origenX'] = $significado->getOrigenX();
62                      $dimensiones[$significado->getCodigo()]['origenY'] = $significado->getOrigenY();
63                      $dimensiones[$significado->getCodigo()]['graficos'] = $significado->getTiposGraficosArray();
64                  }
65              }
66              $rangos_alertas_aux = array();
67              foreach ($fichaTec->getAlertas() as $k => $rango)
68              {
69                  $rangos_alertas_aux[$rango->getLimiteSuperior()]['limite_sup'] = $rango->getLimiteSuperior();
70                  $rangos_alertas_aux[$rango->getLimiteSuperior()]['limite_inf'] = $rango->getLimiteInferior();
71                  $rangos_alertas_aux[$rango->getLimiteSuperior()]['color'] = $rango->getColor()->getCodigo();
72                  $rangos_alertas_aux[$rango->getLimiteSuperior()]['comentario'] = $rango->getComentario();
73              }
```

En este método se obtiene todos los datos del indicador, como son: Máximo valor del indicado, alertas, opciones de filtrado, ficha técnica, SQL, tabla de datos, fecha de última lectura, entre otras.

Si la respuesta es verdadera se llama la función **dibujarControles(zona_g, resp);** para crear el menú y posteriormente llama a la función **dibujarGrafico(zona_g, $('#' + zona_g + ' .dimensiones').val());** para crear el gráfico.

function dibujarControles(zona, datos): Esta función crea el menú y contenido para cada gráfica. Recibe el id de la zona donde se agrega y los datos para construir el menu.

function dibujarGrafico(zona, dimension): Recibe como parámetros la zona donde se dibujara el grafico y los filtros con los que se dibujara. Llama la función con Ajax del controlador **indicadorController.php** que devuelve los datos del indicador con sus filtros.

$.getJSON(Routing.generate('indicador_datos',{id: $('#' + zona + ' .titulo_indicador').attr('data-id'), dimension: dimension}),

```
/**
 * @Route("/indicador/datos/{id}/{dimension}", name="indicador_datos", options={"expose"=true})
 */
public function getDatos(FichaTecnica $fichaTec, $dimension)
{
    $resp = array();
    $filtro = $this->getRequest()->get('filtro');
    $verSql = ($this->getRequest()->get('ver_sql') == 'true') ? true : false;
    $fechas = $this->getRequest()->get('filtrofecha');

    if ($filtro == null or $filtro == '')
        $filtros = null;
    else
    {

        $filtrObj = json_decode($filtro);
        foreach ($filtrObj as $f)
        {
            $filtros_dimensiones[] = $f->codigo;
            $filtros_valores[] = $f->valor;
        }
        $filtros = array_combine($filtros_dimensiones, $filtros_valores);
    }

    $em = $this->getDoctrine()->getManager();

    $fichaRepository = $em->getRepository('IndicadoresBundle:FichaTecnica');

    $user = $this->container->get('security.context')->getToken()->getUser();

    $fichaRepository->crearIndicador($fichaTec, $dimension, $filtros);
    $resp['datos'] = $fichaRepository->calcularIndicador($fichaTec, $dimension, $filtros, $verSql, $fechas, $user->getId());
    $response = new Response(json_encode($resp));
    if ($this->get('kernel')->getEnvironment() != 'dev')
        $response->setMaxAge($this->container->getParameter('indicador_cache_consulta'));

    return $response;
}
```

Si la respuesta es verdadera se llama a la función **dibujarGraficoPrincipal(zona, $('#' + zona + ' .tipo_grafico_principal').val());** para crear el grafico con el tipo seleccionado.

function dibujarGraficoPrincipal(zona, tipo): Recibe como parámetro el id de la zona y el tipo de gráfico, si este viene vacío por default toma *columnas* y crea una instancia de crear gráfico.

var grafico = crearGraficoObj(zona, tipo); que es el encargado de dibujar el tipo de gráfico.

function crearGraficoObj(zona, tipo): Devuelve una instancia del tipo de grafico seleccionado.

function limpiarZona(zona): Limpia la zona del área de grafico seleccionada donde se cargara un indicador.

function acciones_button(): Se encarga de incluir los eventos de los botones del menú de cada gráfica.

15. GLOSARIO

EL siguiente es una lista de los comandos que nos ayudan a simplificar algunos procesos al momento de trabajar con el desarrollo. Todos se hacen desde la consola.

php app/check.php : Muestra requisitos faltantes para el correcto funcionamiento de symfony2.

php app/console : En lista los comandos que se pueden usar

php app/console cache:clear : Borrar cache (Es más seguro eliminado el contenido de la carpeta cache)

Ejemplo del commando php app/console cache:clear --env=prod --no-debug

php app/console generate:bundle : Crear un nuevo Bundle

php app/console assets:install : Publicar Bundle en el directorio web (Si modificamos algún archivo de la carpeta resources es necesario ir a la consola y ejecutar este comando)

php app/console doctrine:generate:entity : Genera una nueva entity con sus respectivos parametros annotation y sus metodos setter y getter ademas tiene una opcion para generar el repository.

Ejemplos:

php app/console doctrine:generate:entities MINSAL: Para todas las entities

php app/console doctrine:generate:entities MINSAL/IndicadoresBundle/Entity/Boletin : Para una entity en particular

php app/console doctrine:database:create : Crear base de datos

php app/console doctrine:schema:create : Crea el esquema de la base de datos

php app/console doctrine:schema:update –force : Actualiza el esquema de la base de datos por si agregamos mas campos a la entity o modificamos uno.

Para ver las sentencias sql a ejecutar poner --dump-sql al final

php app/console doctrine:generate:crud : Generar el list, edit, new de un proyecto (Para este proyecto no aplica ya que se está manejando sonata_admin para manejar el CRUD).

php app/console generate:doctrine:form AcmeBlogBundle:Post : generar un formulario.

psql -d BASE_A_RESPALDAR -f PATH_ABSOLUTO/SCRIPT.SQL" -U USUARIO –W: Puede restablecer una copia de un dump que se tenga de la base de datos.

-d Nombre de la base

-f Path del archivo a ejecuta

-U usuario

-W clave del usuario

psql -d indicadores -f C:\etab.backup -U admin

php composer.phar create-project symfony/framework-standard-edition path/ Carpeta : Crea un Proyecto symphony 2 nuevo en la ruta que pongamos en lugar de **Carpeta.**

Php app/console doctrine:mapping:convert xml ./src/Acme/BlogBundle/Resources/config/doctrine/metadata/orm --from-database –force : Si ya tenemos una base de datos con este comando nos genera dos archivos en el directorio especificado el cual nos servira en el siguiente paso para generar las entities.

php app/console doctrine:mapping:import NombreBundle annotation: Se especifica cómo queremos que se creen las entities en este caso con annotation.

php app/console doctrine:generate:entities NombreBundle : Proseguimos a generar todas las entitie, con sus respectivos metodos setter y getter. Este metodo genera también todas las relaciones que por default tenga la base de datos.

php app/console fos:js-routing:debug : Debug de rutas, Si por algún motive anexamos más rutas con los parametros annotations y estos nos muestra un error. Para validar que exista la ruta ejecutamos este parámetro que nos devuelve la lista de las rutas creadas.

16. BIBLIOGRÁFIA.

http://librosweb.es/symfony_2_4/capitulo_1.html Libro oficial de symfony 2

http://symfony.es/bundles/ Página con todos los Bundles de symfony 2

http://symfony.es/ Página oficial de symfony 2

https://github.com/checherman/etab Código fuente del etab con su respectivo versionado.