SERVICIOS

Servidor Web

Es el principal en Chil 2.0 (ch2), está desplegado en el servidor IIS del 138.100.136.139.

Este servicio es el encargado de servir páginas web y de toda la interacción con el usuario final.

En la carpeta **c:\inetpub\wwwroot** existen 4 directorios, uno para cada uno de los sitios que tenemos actualmente: chil.me (ch2), www.cepesca.es (cepesca), www.agripa.org (agripa) y www.recopes.es (recopes). El código fuente de las 4 carpetas es exactamente el mismo, solo varía el fichero de configuración (**web.config**) y el fichero **favicon.ico**, ambos se encuentran en la raíz de cada carpeta.

Archivo de configuración (web.config)

A continuación se detallan las claves más importantes fichero de configuración:

Clave	Descripción	Ejemplo
app.anonymous_id	Es el id del usuario anónimo en la BD. Es el mismo en todos los ficheros de configuración	1
app.site_url	Es la url del sitio	En el caso de chil.me es http://www.chil.me. En el caso de agripa es http://www.agripa.org y así sucesivamente
app.piwik.enabled	Esta clave se usa para habilitar/inhabilitar el uso de Piwik, admite True o False	Por el momento está en False en todos los ficheros pues no está funcionando el Piwik
app.images_server	Es la url del servidor de imágenes	En todos los casos es http://chilmedia.org
app.widgets.path	Es la ruta física donde se encuentra el código Liquid para renderizar los componentes de manera standard	Para todos los casos está en C:\inetpub\wwwroot\Ch2\Widgets
app.host	Es parecido al app.site_url	Ejemplo: http://agripa.org

app.subdomain_format	Es el formato que siguen los subdominios en los diferentes sitios	En el caso de chil.me es http://{0}.chil.me, en el caso de agripa es http://{0}.agripa.org
app.default_site	Es el sitio web por defecto que debe aparecer en caso de que acceda por una url distinta a chil.me. El nombre debe coincidir con el campo identifier del actor en la BD (agripa, recopes, cepesca)	En el caso de agripa sería agripa, En el caso de chil.me esta clave no existiría porque no hay sitio por defecto.
app.subdomain_length	No se usa mucho, normalmente es para cuando se accede a los sitios vía IP	Normalmente 2
app.mail.sender.address	Es la dirección de correo del sender que aparece al enviar correos desde chil a los usuarios	Ejemplo: administrator@chil.org
app.mail.sender.display_name	Nombre que aparece cuando se envía un correo desde chil	Ejemplo: Agripa
app.mail.templates_path	Apunta a la dirección donde están las plantillas en código Liquid para el envío de correos	Normalmente ~/MailTemplates
app.mail.enabled	Indica si está habilitado (True) o no (False) el correo electrónico	Normalmente True
app.tagging.url	Es la url del sistema de tagging de fernando	Normalmente 138.100.136.137
app.tagging.enabled	Indica si está habilitado (True) o no (False) el sistema de tagging	Normalmente True
app.tagging.default_site_id	Es el id del sitio que aparece en el sistema de tagging de fernando	En el caso de agripa el id es 4. Por el momento es donde único se ha usado
app.schedule_tasks.enabled	Indica si están habilitadas (True) o no (False) las tareas en background	Normalmente True
app.searching.enabled	Indica si están habilitado	Normalmente True

	(True) o no (False) el servicio de búsquedas de texto completo	
app.searching.host	Es la url del servidor de búsquedas	Normalmente http://138.100.136.139:9200
app.searching.index	No se usa mucho, es para poner un sufijo a los índices (que son como las tablas) al servidor de búsquedas seguido del id del sitio	Normalmente ch2
app.analytics.enabled	Indica si están habilitado (True) o no (False) un servicio propio de analytics que se estubo desarollando en NodeJs	
app.analytics.service_url	Url del servicio de analytics propio	Normalmente http://138.100.136.140:3001
quartz.scheduler.instance_name	Nombre del servidor quartz para procesos en ebackground	Normalmente ServerScheduler
quartz.scheduler.proxy	Leer documentación de quartz	Poner en true
quartz.thread_pool.thread_count	Leer documentación de quartz	Poner en 0
quartz.scheduler.proxy.address	Es la url del servidor quartz	Normalmente tcp://localhost:556/QuartzScheduler
fb.AppId	Es el Id de la app de facebook. Debe existir una por cada sitio. Sirve para hacer login con facebook	
fb.AppSecret	Contraseña de la app de facebook	
fb.RediretUrl	Donde te debe redirigir facebook luego de hacer login. Debe coincidir con lo que se especifica en la app de facebook si no se produce un error	Ejemplo: http://agripa.org/login/facebook
google.api_key	Es la clave que se usa para	AIzaSyBinWiTMFgQuCiYID

	poder hacer login con google. Por el momento es la misma para todos los sitios	sIkggkCV36iZ25rN8
foursquare.ClientId	Es el id de la app de foursquare. Es el mismo para todos los sitios. Se usa para el dialogo de “lugares cercanos” cuando se va a escoger una localización	0V4FVOICNUWWGVSSMAHUL1F3SCP3V1J4MOSP3UPEMMDNXL4E
foursquare.ClientSecret	Contraseña de la app de foursquare	2M0MMQKKKU3KYNZHQJWAMMGBASSRVDJE3LAUCVNIB4H5SKL5

Aparte de estas claves personalizadas que se en encuentran en la sección “appSettings” del web.config, existen otras que se describen a continuación:

En la sección “connectionStrings” se especifica la url del servidor de bases de datos bajo la clave Db, ejemplo:
<add name="Db" connectionString="data source=138.100.136.138;Persist Security Info=True;Initial Catalog=Ch2;User ID=sa;Password=xyz;" providerName="System.Data.SqlClient" />

En la sección system.web, en la etiqueta “**httpCookies**”, se especifica el dominio que van a tener las cookies. Si esto no se hace inicializa adecuadamente se pierde el usuario autenticado en los diferentes subdominios. En el caso de agripa queda de la siguiente forma:
<httpCookies domain=".agripa.org" />. Por su parte en el caso de Chil.me queda como ***<httpCookies domain=".chil.me" />***

En la etiqueta **system.net/mailSettings** se encuentra la etiqueta **network** que es la que se usa para setear las variables del servidor de correos, normalmente se define de la siguiente forma:
<network host="smtp.mandrillapp.com" port="587" userName="chil.social@gmail.com" password="LYjLtN7MBCX5LctlKim4Sw" defaultCredentials="false"/>

La sección **log4net** se usa para los logs de la aplicación, lo más importante es la etiqueta “file” que se usa para especificar la ruta donde se guardan los logs, normalmente tiene el siguiente valor:
<file value="logs\\ch.log"/>
Con esto se indica que los logs están en la carpeta logs del sitio

Carpetas del sitio:

Nombre	Descripción
App_Data	Esta carpeta solo contiene un archivo con una base de datos de geolocalización (GeoLiteCity.dat)
Backups	Se usa para guardar distintas versiones del sitio. Antes de subir un cambio normalmente se hace un Backup de lo que hay funcionando en ese momento y se salva en esta carpeta
bin	Es junto con Views una de las carpetas fundamentales. Contiene todas las dll necesarias para que funcione el sitio. Las dll propias de Chil comienzan con el prefijo App., ejemplo: App.Web.Mvc.dll, App.Services.dll
Content	En esta carpeta se almacenan los CSS necesarios para que funcione el sitio
Fonts	Fuentes del sitio
HtmlTemplates	Se guardan algunas plantillas necesarias para que funcione algunas pantallas de administración. La idea es tratar de eliminar esta carpeta en el futuro
Images	Aquí están todas las imágenes del sitio. La idea aquí es usarla cada vez menos y llevarlas al servidor de imagenes o utilizar archivos de Fonts
logs	Aquí se guardan logs del sistema. por cada día se guarda un log fundamentalmente con errores de ejecución etc
MailTemplates	Esta carpeta contiene archivos escritos usando la sintaxis de Liquid necesarios para formatear los correos electrónicos que se envian a los usuarios
ReactComponents	Esta carpeta es nueva y se espera que en un futuro sustituya a las carpetas Views y Scripts. Actualmente contiene código en JSX para que funcione la administración de productos y de posts. La idea es que todas las pantallas de administración de Chil se haga usando tecnología React y asi lograr una SPA (Single Page Application)
Scripts	Contiene los scripts de la aplicación. Internamete losscripts fndamentales están en la carpeta App, las demas carpetas son librerías externas o algunas de reciente creación como Angular para encapsular scrips de AngularJs o Api para encapsular todas las llamadas API y poder lograr una SPA

StandardWidgetsTemplates	En esta carpeta se almacenan ficheros en código Liquid necesarios para pintar los componenetes standard de la aplicación. Por ejemplo cuando en una página e pone un componente de tipo Posts y en la vista del mismo aparece standard/posts, el engine de renderizado de páginas va a esta carpeta y dentro de esta carpeta busca otra llamada posts y así sucesivamente hasta que se encuentra el fichero a usar
Styles	Contiene CSS útiles sobre todo para las páginas web de sitios web como agripa, cepesca, agricola-cafe, etc

Servidor de imagenes

El otro servicio del stack es el servidor de imágenes que está desplegado en el servidor IIS del 138.100.136.140 en la carpeta **d:\Ch2ImgServer** bajo el nombre de **ch2imagesserver**. Este servidor también sirve algunas imagenes para el Chil 1.0

Las claves más importante de este servicio son las cadenas de conexión con las bases de datos y las rutas.

En el caso de las cadenas de conexión, existe una para chil 1 y una para chil 2 en la sección **connectionStrings** del **web.config** y queda de la siguiente manera:

Para el caso de chil 2, la configuración es la siguiente:
<add name="Db" connectionString="data source=138.100.136.138;Persist Security Info=True;Initial Catalog=Ch2;User ID=sa;Password=xyz;" providerName="System.Data.SqlClient" />

Por su parte para la cadena de conexión de chil 1 es la siguiente:
<add name="Ch1" connectionString="data source=138.100.136.138;Persist Security Info=True;Initial Catalog=Chil;User ID=sa;Password=xyz;" providerName="System.Data.SqlClient" />

Las rutas están definidas en la sección **resizer/plugins**. Actualmente se cuenta con las siguientes:

<add name="SqlReader" prefix="~/" connectionString="Db" idType="BigInt" blobQuery="SELECT Content FROM Media WHERE Id=@id" modifiedQuery="SELECT CreationDate as ModifiedDate, CreationDate as CreatedDate From Media WHERE Id=@id" existsQuery="SELECT COUNT(Id) FROM Media WHERE Id=@id" cacheUnmodifiedFiles="true" />

Esta es la ruta standard, que en un futuro debe desaparecer porque expone el id de de las imágenes, ejemplo: http://chilmedia.org/20723.jpg. Sirve para imagenes de Chil 2, esto se

puede ver en la clave **connectionString** que apunta a “Db” que es la base de datos de Ch2, a continuación admite un número entero largo (ejemplo 20723). Para recuperar la imagen de la base de datos usa la consulta ***SELECT Content FROM Media WHERE Id=@id*** sustituyendo ***@id*** por el valor que se pasa en la URL (en este caso 20723) y se devuelve la imagen en formato jpg.

<add name="SqlReader" prefix="~/v1/" connectionString="Ch1" idType="Int" blobQuery="SELECT FileContent FROM Media WHERE MediaId=@id" modifiedQuery="SELECT CreatedOn as ModifiedDate, CreatedOn as CreatedDate From Media WHERE MediaId=@id" existsQuery="SELECT COUNT(MediaId) FROM Media WHERE MediaId=@id" cacheUnmodifiedFiles="true" />

Se usa para las imágenes de Chil 1. Ejemplo: http://chilmedia.org/v1/18240.jpg. Notar el prefijo ***/v1/*** en la url. Opera de manera similar al descrito anteriormente pero haciendo uso de la base de datos de ch1 (***connectionString="Ch1"***) y con una consulta a la base datos diferente: ***SELECT FileContent FROM Media WHERE MediaId=@id***

<add name="SqlReader" prefix="~/v2/file-preview/" connectionString="Db" idType="UniqueIdentifier" blobQuery="SELECT m.Content FROM [File] f INNER JOIN [Media] m ON f.PreviewId = m.Id WHERE f.Uuid = @id" modifiedQuery="SELECT f.CreationDate as ModifiedDate, f.CreationDate as CreatedDate FROM [File] f INNER JOIN [Media] m ON f.PreviewId = m.Id WHERE f.Uuid = @id" existsQuery="SELECT COUNT(f.Id) FROM [File] f INNER JOIN [Media] m ON f.PreviewId = m.Id WHERE f.Uuid = @id" cacheUnmodifiedFiles="true" />

Se usa para vista previa de archivos de chil2, es necesario pasarle el uid del fichero. Debe eliminarse en un futuro y sustituirla por /v2/media/ que se detalla a continuación

<add name="SqlReader" prefix="~/v2/media/" connectionString="Db" idType="UniqueIdentifier" blobQuery="SELECT Content FROM Media WHERE RowGuid=@id" modifiedQuery="SELECT CreationDate as ModifiedDate, CreationDate as CreatedDate From Media WHERE RowGuid=@id" existsQuery="SELECT COUNT(Id) FROM Media WHERE RowGuid=@id" cacheUnmodifiedFiles="true" />

Es la forma más genérica y segura de acceder a las imágenes de Ch2. Ejemplo: http://chilmedia.org/v2/media/14809d0d-0709-4714-a5a0-435cc76d652b.jpg
Acepta un parámetro de tipo Guid (***idType="UniqueIdentifier"***) y ejecuta una consulta sobre la base de datos de ch2 (***connectionString="Db"***) concretamente sobre la tabla Media (***SELECT Content FROM Media WHERE RowGuid=@id***)

A las url de las imágenes se les puede pasar parámetros para cabiarlas de tamaño, escalarlas etc, (ej: http://chilmedia.org/v2/media/14809d0d-0709-4714-a5a0-435cc76d652b.jpg?w=200) para una explicación de todos los parámetros que se le pueden pasar a las url, ver la documentación oficial del sitio en http://imageresizing.net/

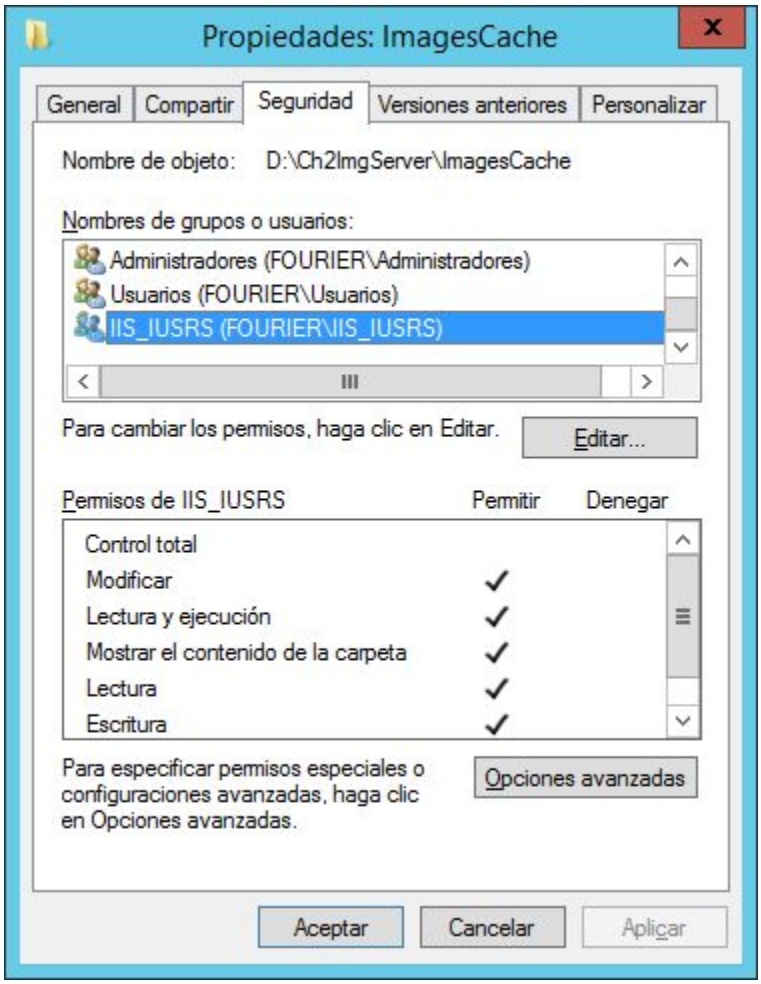
Todas las imágenes que se procesan son almacenadas en una caché que se encuentra en disco duro. Esto se usa para no tener que procesar varias veces la imagen o para no tener que ir a la base de datos a buscarla.

En el fichero de configuración existe una clave que se llama **diskCache** donde se configura la carpeta donde se van a almacenar estas imágenes.

<diskCache dir="~/ImagesCache" subfolders="128" />

Con esto estamos indicando que la carpeta ImagesCache del sitio se usará como caché de imágenes y que internamente manejará 128 subcarpetas para almacenar las imágenes procesadas. Estas subcarpetas se crean automáticamente y si se borran no pasa nada, la que tiene que estar es la carpeta **ImagesCache**.

Es importante tener en cuenta que el usuario IIS_IUSRS debe tener permisos para escribir en esta carpeta si no se produce un error.

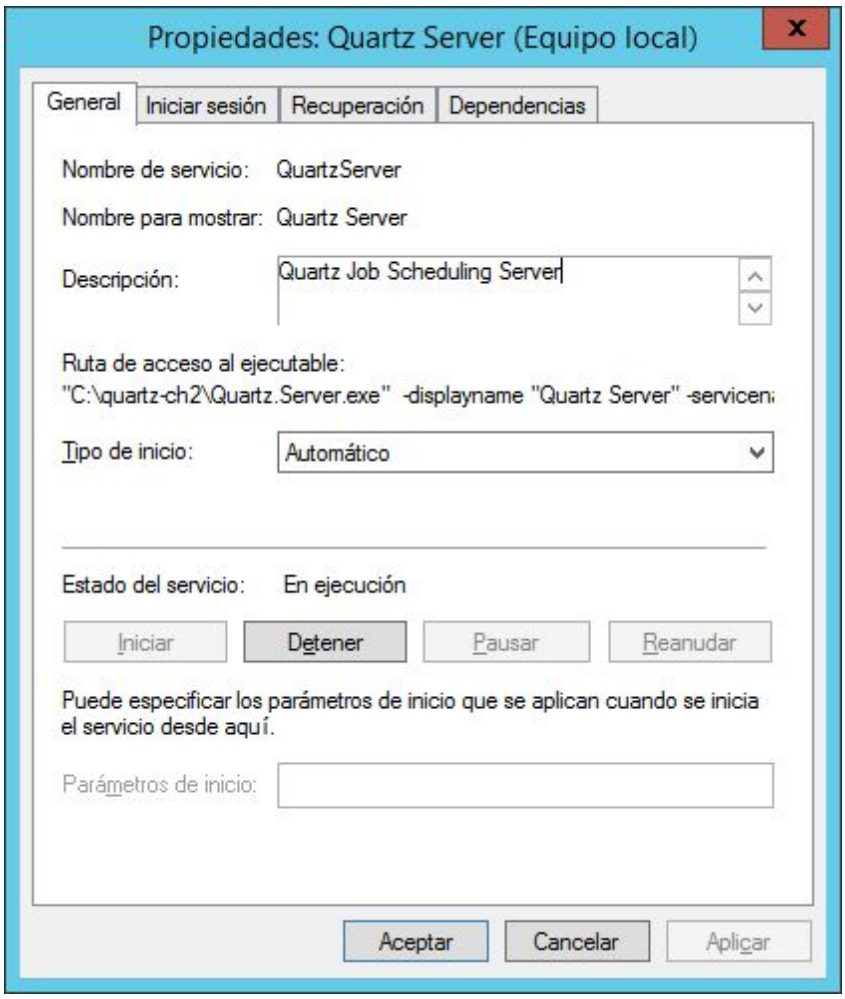


Servidor Quartz

Este servidor se usa para levantar procesos en background. Se encuentra corriendo en el servidor 138.100.136.138, en la carpeta **c:\quartz-ch2**. Actualmente se usa para 2 cosas:

1. Pasar datos del servidor de bases de datos relacional (Sql-Server) al servidor de búsqueda de texto completo (ElasticSearch). Este proceso se ejecuta cada una hora y es necesario ejecutar unos scripts para indexar cada sitio de manera independiente
2. Para poner un post como publicado. A los posts que están en *draft* o *waiting_for_approval* se les puede planificar una determinada hora a la cual se deben poner como *published* y esto se logra con el servidor Quartz.

Quartz actualmente corre como un servicio de windows como se muestra en la siguiente imagen



Adicionalmente Quartz está configurado para que use SqlServer como almacén de jobs, esto se configura en el archivo C:\quartz-ch2\quartz.config, las claves fundamentales de este archivo son las siguientes:

Clave	Descripción

quartz.scheduler.exporter.port = 556	Indica que el sericio de Quartz escucha peticiones en el puerto 556
quartz.scheduler.exporter.channelType = tcp	Canal que se usa, en este caso TCP
quartz.jobStore.type = Quartz.Impl.AdoJobStore.JobStoreTX, Quartz	Indica que va a usar un store que se manejará con ADO.NET (aka base de datos relacional)
quartz.dataSource.default.connectionStringName = quartz	Indica el nombre de la cadena de conexión del archivo Quartz.Server.exe.config que se usará para comunicarse con la base de datos

Existe también un archivo llamado Quartz.Server.exe.config que contiene la configuración de las cadenas de base de datos, etc

<add name="quartz" connectionString="data source=138.100.136.138;Persist Security Info=True;Initial Catalog=Ch2Quartz;User ID=sa;Password=xyz;" providerName="System.Data.SqlClient" />
Esta es la cadena de conexión que se usará la base de datos de quartz, donde almacenarán todos los triggers, jobs etc propios de Quartz. Notar que se llama **quartz** como se especificó en el archivo anterior (quartz.config)

<add name="Db" connectionString="data source=138.100.136.138;Persist Security Info=True;Initial Catalog=Ch2;User ID=sa;Password=xyz;" providerName="System.Data.SqlClient" />
Esta es la cadena de conexión que usan las dll de ch2 para su funcionamiento

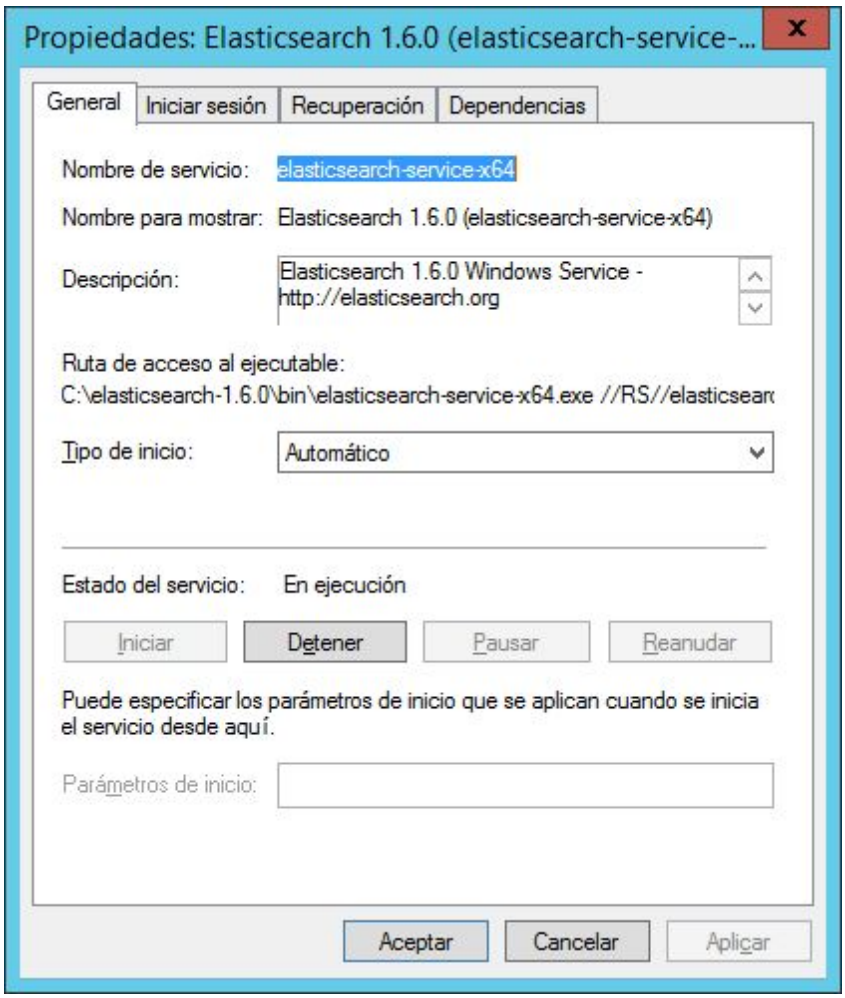
<appSettings>
<add key="app.searching.host" value="http://138.100.136.139:9200" />
</appSettings>
Url del servidor ElasticSearch. Es necesario especificarla pues una de las tareas de Quartz que se ejecutan actualmente es “alimentar” este servidor a partir de los datos del servidor de base de datos como se explicó anteriormente.

Para ver como se instala Quartz como servicio se puede leer el siguiente post: http://geekswithblogs.net/TarunArora/archive/2012/11/16/install-quartz.net-as-a-windows-service-and-test-installation.aspx

La documentación oficial de Quartz está en http://www.quartz-scheduler.net/

Servidor de Búsquedas

El servidor de búsquedas que se usa es el ElasticSearch 1.6 (https://www.elastic.co/). El servidor se encuentra corriendo en el servidor 138.100.136.139:9200

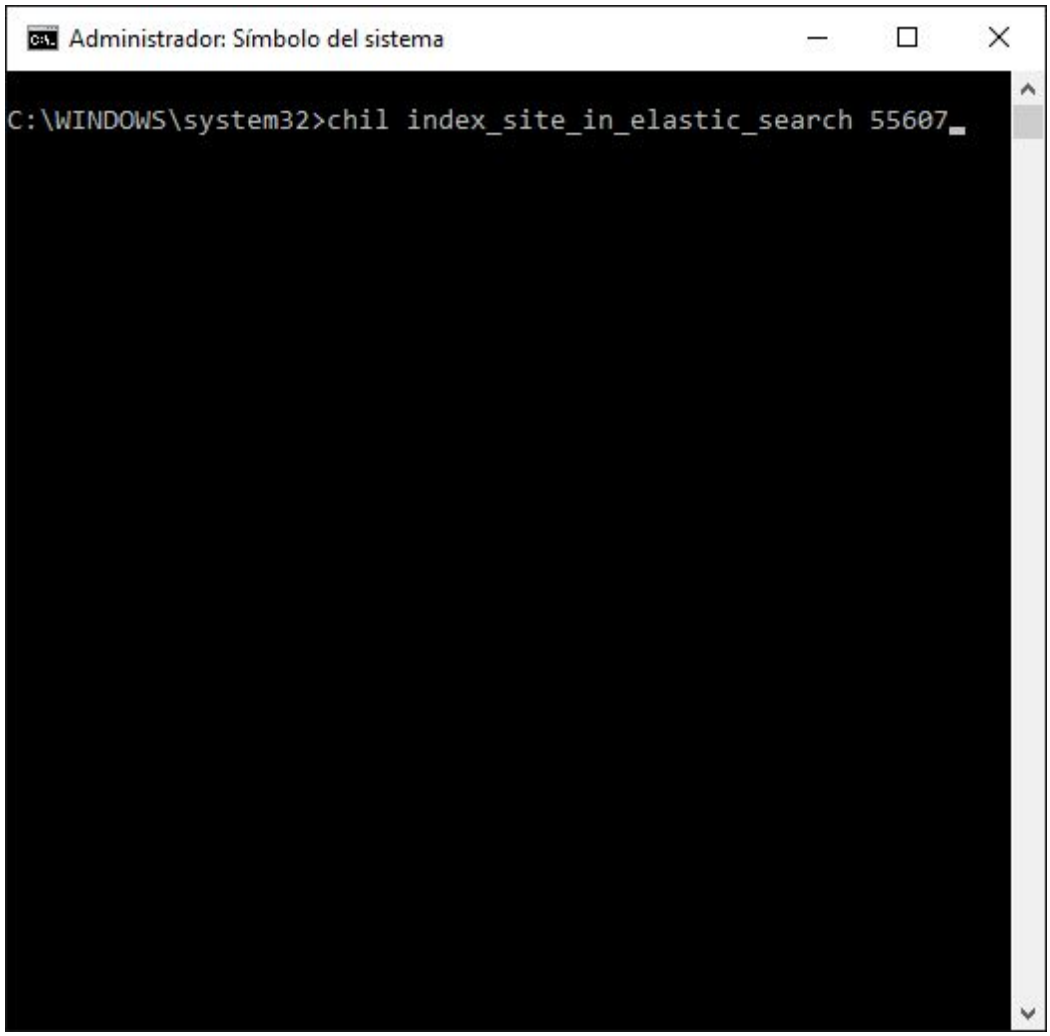


Se encuentra en la carpeta ***c:\elasticsearch-1.6.0***

El servidor cuenta de una interfaz REST para administrarlo. En nuestro caso por cada sitio se se crea un índice que es como una base de datos y en cada índice se le ponenlos documentos, que son como las tuplas de una tabla, con los datos que se quieren indexar.

Antes que todo es necesario indexar un sitio por primera vez y luego crear un Job en Quartz para que cada una hora se actualice dicho índice.

Para indexar un sitio es necesario correr desde la consola de administración el comando ***index_site_in_elastic_search*** seguido del id que tiene el sitio en la base de datos como se muestra a continuación:



Con este comando estamos indexando el sitio AproGip en el servidor de ElasticSearch.

Si lo que queremos es que el sitio se indexe de forma automáticamente es necesario ejecutar el comando de chil schedule_elastic_search_index seguido del id del sitio, ejemplo: ***chil schedule_elastic_search_index 55607***, con esto le decimos al servidor Quartz que cada una hora indexel el sitio 55607 que en este caso se corresponde con AproGip.

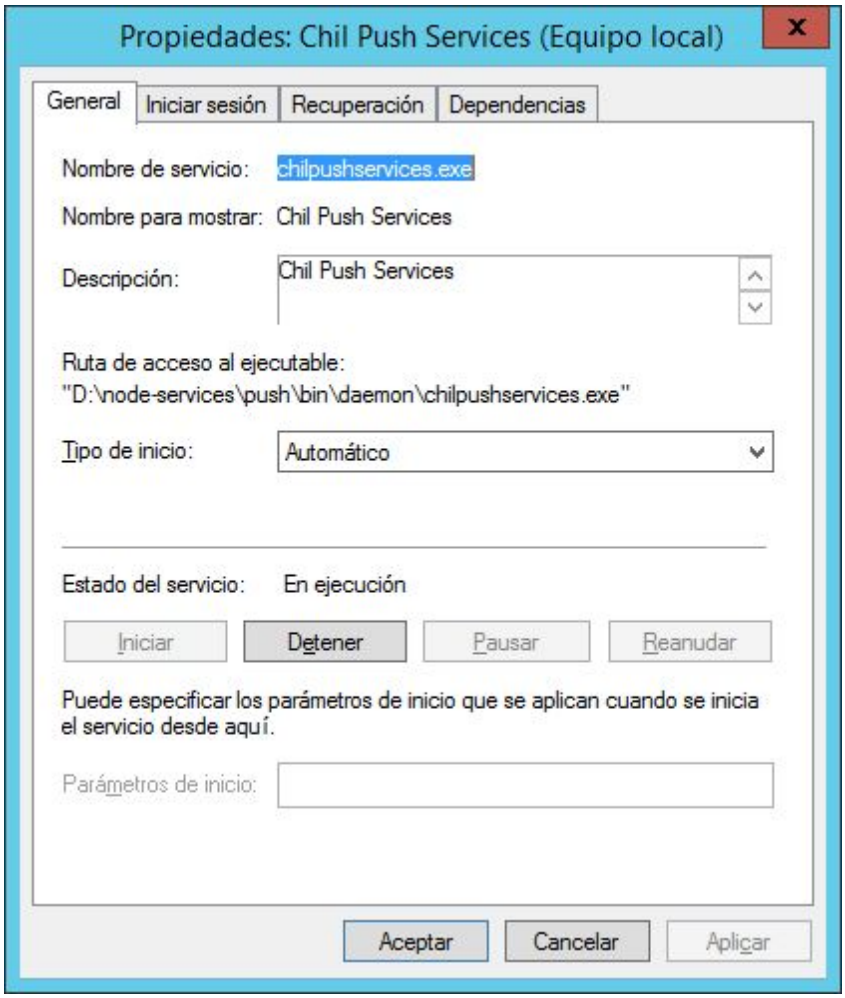
Servidor Push

Este servidor está escrito en NodeJs y está corriendo en la IP http://138.100.136.140:3002 en la carpeta ***d:\node-services\push***.

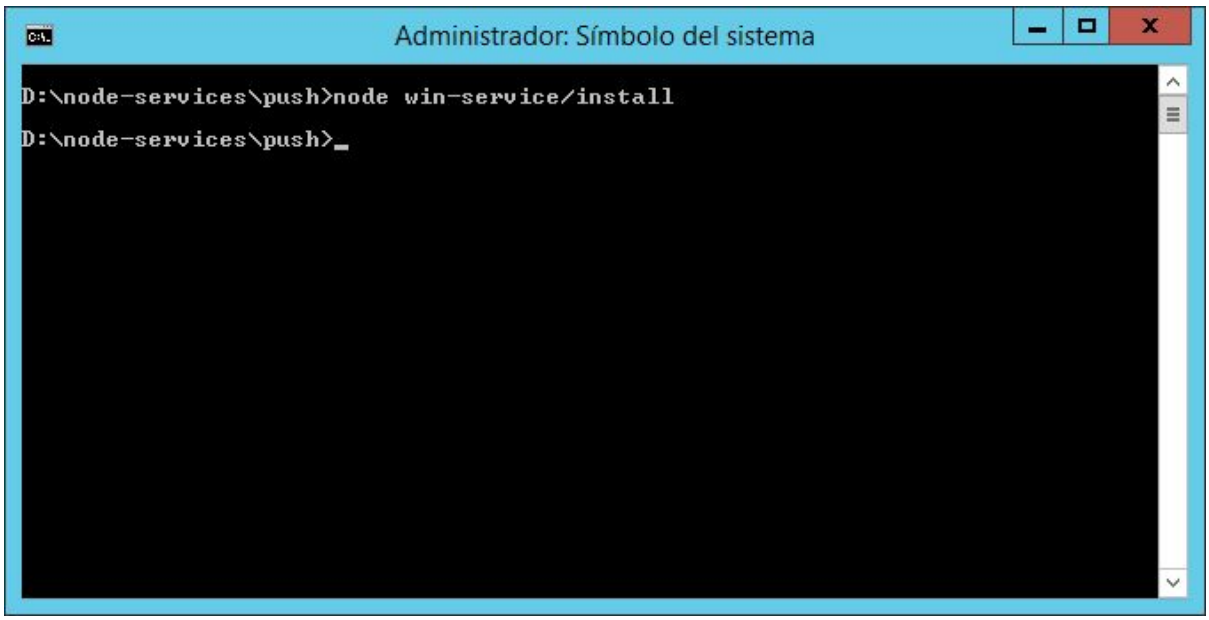
Para que el servicio corra es necesario instalar NodeJs en el servidor. Si se desea comprobar que el servicio está corriendo de forma correcta ejecutar la siguiente solicitud: http://138.100.136.140:3002/ping y se debe obtener una respuesta como la siguiente:

{"ok":true,"pong":"smile, the Chil Push Service is running"}

El servicio se encuentra corriendo como un servicio de Windows como se muestra en la siguiente imagen:



Para crear este servicio es necesario pararse en la ruta donde está el código del servicio y luego ejecutar el comando ***node win-sercice/install*** como se muestra en la siguiente imagen



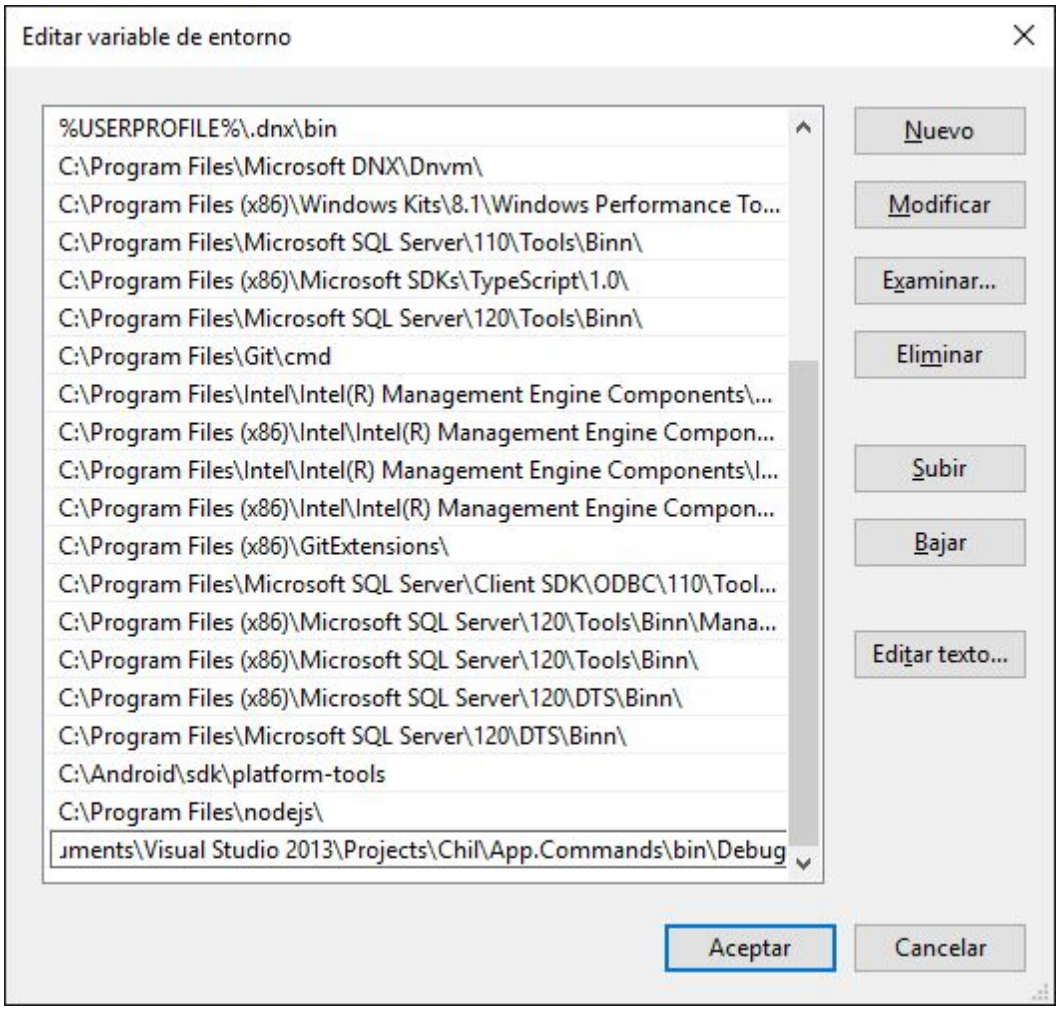
Por el momento este servicio solo se comunica con canales de GCM por tanto existe un fichero de configuración en la raíz de la carpeta llamado **config.json** que entre otras cosas el parámetro más importante es el ***gcm.apikey*** donde se especifica la llave que se obtiene en el proceso de creación de la aplicación en la consola de Google. Si se desea cambiar esta llave es necesario parar el servicio, cambiar la llave y luego reiniciar el servicio

Servidor Analytics

Es un servidor escrito en NodeJs qus se estaba desarrollando pero actualmente no funciona pues MongoDb se detiene inesperadamente en Windows y no he posido averiguar porqué

COMANDOS

Existe una utilidad de reciente creación para ejecutar comandos, el proyecto asociado a esto es ***App.Commands***. Como recomendación, en las variables del sistema poner una nueva entrada al lugar donde se encuentra el código de este proyecto de manera que sea sencillo tener acceso al comando ***chil*** como se muestra en la siguiente imagen



Actualmente los comandos que existen son los siguientes:

Comando	Parámetros	Ejemplo
reset_password Resetear el password de un usuario	email new_password	chil reset_password h@chil.org newPsw
index_site_in_elastic_search Indexar un sitio en el servidor de búsquedas de texto completo	site_id	chil index_site_in_elastic_search 55607
schedule_elastic_search_index Indexar sitio automáticamente cada una hora	site_id	chil schedule_elastic_search_index 55607

En el fichero de configuración de este proyecto lo más importante es lo siguiente:

<connectionStrings>
<add name="Db" connectionString="data source=138.100.136.138;Persist Security Info=True;Initial Catalog=Ch2;User ID=sa;Password=axyz;" providerName="System.Data.SqlClient" />
</connectionStrings>
Aquí se especifica la cadena de conexión a la base de datos de chil

<add key="quartz.scheduler.proxy.address" value="tcp://138.100.136.139:556/QuartzScheduler" />
URL del servidor Quartz

<add key="app.searching.host" value="http://138.100.136.139:9200" />
URL del servidor ElasticSearch

HERRAMIENTAS, FRAMEWORKS y LIBRERíAS

- Para la aplicación Web se usa ***ASP.NET MVC***
- Para la construcción del API se usa ***ASP.NET WebAPI***
- Como motor de inyección de dependencias se usa ***Unity*** y ***ServiceLocator***
- Para renderizar las páginas de administración del sitio etc se usa ***Razor***
- Para la comunicación con el servidor de base de datos se usa ***EntityFramework 6.0*** aunque en la actualidad se está migrando a ***Dapper***
- La nueva versión de las administración de Posts y la administración de Productos está hecha con ***ReactJs 1.4***
- Para el renderizado de las páginas de usuario se usa ***DotLiquid***
- En el front-end lo que se usa de forma casi exclusiva es ***JQuery***

ESTRUCTURA DEL CÓDIGO

Proyecto	Descripción
App.Analytics	En este momento no se usa. Su función es hacer de puente entre el código de la aplicación web y el servicio de Analytics
App.ConsoleTests	En este proyecto solo existen pruebas que se hacen en momentos puntuales antes. También contiene código de migración de datos de Ch1 a Ch2
App.Core	Define estructuras genéricas para manejar seguridad, clases abstrractas, etc. Lo fundamental es una carpeta que se llama Domain que contiene la definición del dominio con la clase Actor etc, en la actualidad todo ese código se está migrando a App.Services
App.Core.Web	Define clases genéricas y utilidades para proyectos Web. Aque se definen, entre otras, la clase ***AppController*** de la cual heredan todos los controladores de la aplicación, la clase ***RazorWebViewPage*** de la cual heredan todas las vistas y la clase ***IoCControllerFactory*** que se usa para construir los controladores
App.CustomTags	Aquí se definen la lógica para renderizar y captar tags personalizados que los usuarios pueden insertar, por ejemplo, en el body de los posts, ejemplo: [[twitter status='http://…….']]. Todos los tags personalizados deben implementar la interfaz ***ICustomTagTransformer***
App.DotLiquidExtensions	Define extensiones que se le pueden hacer al lenguaje Liquid (usado para renderizar las páginas web de los sitios). Fundamentalmente define la implementacion de las etiquitas ***{%js%} {%css%} {%widget%}{%endwidget%}*** así com varios filtros
App.Dto	Define clases que se usan para devolverselas al cliente (aplicación web o WebAPI). En un futuro debe cambiar y almacenar solo los modelos de la aplicación o simplemente debe desaparecer y que los modelos se definan a nivel de cada servicio. Ejemplo de clases que se definen aquí: ***PostDto***, ***Product***, ***CommentDto***, etc
App.Dto.Repositories	Define métodos de comunicación con la base de datos para obtener objetos DTO, en un futuro debe desaparecer totalmente
App.EventHandlers	Define desencadenadores de eventos que se ejecutan cuando sucede alguna acción en el sistema. Ver

	EventSource
App.ImgServer	Es el código del servidor de imágenes
App.Persistence	Define métodos para comunicación con el servidor de bases de datos usando Dapper. En el futuro debe desaparecer y todos los repositorios que se implementan aquí deben estar al nivel de servicio
App.Persistence.Ef	Idem al anterior, solo que en este caso usa EntityFramework
App.Push	Para comunicarse con el servidor Push
App.Queries	Define algunas cosas que no se han podido refactorizar. En el futuro debe desaparecer
App.ScheduledTasks.EventHandlers	Define clases, en respuesta eventos del sistema, para ejecutar procesos en background. Actualmente la fundamental es la clase ***PostEventHandlers*** que se encarga de poner Jobs en Quartz para pasar un post de draft a published. En un futuro debe desaparecer
App.ScheduledTasks.Jobs	En este ensamblado se definen el código de los procesos que deben correr en background en el servidor Quartz. actualmente la clase ***PostPublisherJob*** se encarga de poner un post en estado published y la clase ***IndexSiteJob*** se encarga del proceso de indexar un sitio en el ElasticSearch. Este ensamblado, junto a sus dependencias, debe desplegarse en servidor Quartz
App.Searching	La clase fundamental de este ensamblado es ***ElasticsearchEngine*** que implementa las interfaces ***IIndexer*** encargada de indexar un sitio en ElasticSearch y ***ISearcher*** que se encarga de dado un texto hacer una búsqueda de ese texto en ElasticSearch
App.Searching.Web.Mvc	Este ensamblado define un controlador (***SearchingController***) para usarlo desde la capa Web. No sé si sea conveniente implementar este controlador en App.Web.Mvc directamente, se ha dejado así para dejar claro que es un subsistema aparte.
App.Services	Es el ensamblado fundamental del sistema. Contiene toda la lógica de la aplicación. por cada funcionalidad de la aplicación existe una carpeta que implementa su lógica, ejemplo: Posts, Comments, Pages, etc. Quizás sea buena idea fragmentarlo en varios ensamblados que se puedan mantener y desplegar con mayor facilidad.
App.Utils	Contiene utilidades de todo tipo, por ejemplo para trabajar con colecciones, imágenes, conexiones HTTP, seguridad,

	etc
App.Web.Api	Implementa la API Web del sistema. los controladores de este ensamblado son flat, lo único que hacen es recolectar los datos de la petición e invocar un método del Service. Es posible que en un futuro se solape con App.web.Mvc y se cree solo un proyecto con la API y todo lo demás (renderizado de pantallas etc) corran en el cliente, aka SPA
App.Web.Mvc	Contiene el front de la aplicación, incluyendo las vistas controladores, scripts, etc. Aquí se definen las pantallas con las que interactúa el usuario. Es el proyecto que se despliega en el servidor web
App.Web.Tagging	Contiene controladores para comunicarse con el sistema de tagging
App.Web.Widgets	Define los componentes que se pintan en las páginas web de los usuarios, el más importante es ***PostsStreamWidgetAdapter*** que se encarga de pintar posts

CÓMO SE CREA UN WIDGET?

Para crear un widget es necesario hacer 2 cosas:

1. Implementar la interface

CóMO SE RENDERIZA UNA PÁGINA

EVENT SOURCE

MODELO DE SEGURIDAD

OBJECT BUILDERS

CÓMO SE CREA UN JOB DE QUARTZ