Programación Web

COMPILADORES, GESTORES DE BASES, BASES DE DATOS

Sanada Yukimura – Fernando Ruiz Vidal

TEMAS DE RETROALIMENTACION  INVESTIGACION Y QUIZAS CON AUDIO

# SASS Compilador De CSS.

Los preprocesadores CSS son herramientas para los desarrolladores de sitios web, que permiten traducir un código de hojas de estilo no estándar, específico del preprocesador en cuestión, a un código CSS estándar, entendible por los navegadores.

Los preprocesadores básicamente nos ofrecen diversas utilidades que a día de hoy no se encuentran en el lenguaje CSS, o bien no son compatibles con todos los navegadores. Ejemplos pueden ser variables, anidación de selectores, funciones (denominadas mixins), etc.

Al desarrollar con un preprocesador se consigue principalmente un ahorro de tiempo, ya que tenemos que escribir menos código para hacer las cosas. Pero también conseguimos una mayor facilidad de mantenimiento del código, gracias a una mejor organización del código y la posibilidad de editar una vez ciertos valores y que afecten a decenas, o cientos, de lugares del código CSS generado.

No todo son ventajas cuando hablamos de preprocesadores, ya que nos obligan a aprender un nuevo lenguaje y escribir código no estándar. Ese código no estándar debe ser compilado en CSS, lo que afecta también al flujo de desarrollo. El código del preprocesador no lo entiende el navegador y por lo tanto nos tenemos que dar el trabajo de traducirlo, antes de llevar un sitio web a producción. Afortunadamente, ese flujo está bastante depurado y existen diversas alternativas para la optimización de esa traducción, de modo que no tengamos que intervenir manualmente y nos ahorre tiempo.

la tarea de aprender Sass se divide en dos bloques principales. Primero aprender a trabajar con el código de Sass y conocer su sintaxis especial. La segunda tarea sería aplicar Sass en tu flujo de trabajo, de modo que puedas usar el preprocesador fácilmente y no te quite tiempo el proceso de compilación a CSS. Veremos las dos partes en este primer artículo del Manual de Sass, comenzando por examinar su sintaxis.

Obviamente, no vamos a poder explicar la sintaxis de Sass en unas pocas líneas. Para eso tenemos el manual completo, pero queremos comenzar dando algunos ejemplos para que veas cómo es.

Lo primero que debes conocer de Sass es que existen dos tipos de sintaxis para escribir su código:

* **Sintaxis Sass**: esta sintaxis es un poco diferente de la sintaxis de CSS estándar. No difiere mucho. Por ejemplo, te evita colocar puntos y coma al final de los valores de propiedades. Además, las llaves no se usan y en su lugar se realizan indentados.
* **Sintaxis SCSS**: Es una sintaxis bastante similar a la sintaxis del propio CSS. De hecho, el código CSS es código SCSS válido. Podríamos decir que SCSS es código CSS con algunas cosas extra.

En la práctica, aunque podría ser más rápido escribir con sintaxis Sass, es menos recomendable, porque te aleja más del propio lenguaje CSS. No solo es que te obligue a aprender más cosas, sino que tu código se parecerá menos al de CSS estándar y por lo tanto es normal que como desarrollador te sientas menos "en casa" al usar sintaxis Sass. En cambio, al usar SCSS tienes la ventaja de que tu código CSS de toda la vida será válido para el preprocesador, pudiendo reutilizar más tus conocimientos y posibles códigos de estilo con los que vengas trabajando en los proyectos.

Como entendemos que al usar un preprocesador es preferible mantenerse más cerca del lenguaje CSS, **en este manual usaremos siempre sintaxis SCSS**.

Ahora veamos algunas cosas sobre la sintaxis de Sass. Por ejemplo, el uso de variables.

$color-fondos: #F55;

h1 {

background-color: $color-fondos;

}

En este código estás viendo cómo declaramos una variable y a continuación la usamos en un selector. Obviamente, eso no es un CSS válido, ya que no existen este tipo de declaraciones en el lenguaje.

**Nota:** Hoy CSS ya incorpora variables, aunque se usa otra sintaxis. El problema es que no están disponibles todavía en todos los navegadores, por lo que no es totalmente seguro usarlas, a no ser que implementes PostCSS con CSS Next.

Una vez compilado el código anterior, obtendrías un código estándar como el siguiente:

h1 {

background-color: #F55; }

Veremos más sobre variables y muchas otras utilidades de Sass en breve. Pero ahora preferimos dedicar algo de tiempo para explicarte cómo puedes usar Sass en tu proyecto.

# Typescript Lenguaje Superconjuto De Javascript.

[**TypeScript**](https://www.typescriptlang.org/)**es un superconjunto de JavaScript,** pensado para los tiempos modernos y la programación de aplicaciones en HTML5**, y que viene a suplir una carencia del segundo, que no es otra que la de no estar adaptado a proyectos de gran envergadura.**

La propuesta viene acompañada por sus respectivas extensiones para *Visual Studio*, y de una gran plataforma con paquetes, tutoriales y todo lo que un desarrollador necesita para empezar a picar código.

**Lo cierto es que JavaScript se nutre con buena gana de TypeScript, permitiendo trabajar con este lenguaje como si de Java se tratase,** con clases, y orientado a objetos, lo que permite un mayor control de la capa funcional de la aplicación.

Pero **hay un gran pero**, y es del que quería hablaros.

**Actualmente están floreciendo muchos superconjuntos de otros lenguajes.** La idea es interesante, y suple de forma rápida carencias individuales, haciendo evolucionar la programación hacia los límites más abstractos de la computación.

Para que nos entendamos, un superconjunto de A, llamado B, es un conjunto que define todos los métodos (teoremas, atributos, funciones,..) de A, y además unos nuevos. En programación, entendemos un superconjunto de un lenguaje, como otro lenguaje totalmente compatible con el primero, pero que además ofrece nuevas funcionalidades.

**El problema surge cuando el lenguaje A evoluciona, y quizás adquiera de forma interna funcionalidades que ya tenía el superconjunto B**. En ese caso, al superconjunto B no le queda más remedio de:

* O bien dejar su funcionalidad como está, **existiendo duplicidad de funciones** que a lo mejor hacen lo mismo.
* O bien adaptar su funcionalidad a la nueva, requiriendo un trabajo extra, y que por lo general, acaba con **problemas en la compatibilidad** (el típico «*compatible hasta la versión X del lenguaje A*«).

# CRUD Crear, Leer, Actualizar Y Borrar.

El concepto CRUD está estrechamente vinculado a la gestión de datos digitales. CRUD hace referencia a **un acrónimo** en el que se reúnen las primeras letras de las cuatro operaciones fundamentales de aplicaciones persistentes en sistemas de bases de datos:

* **C**reate (Crear registros)
* **R**ead bzw. **R**etrieve (Leer registros)
* **U**pdate (Actualizar registros)
* **D**elete bzw. **D**estroy (Borrar registros)

En pocas palabras, CRUD resume las funciones requeridas por un usuario para crear y gestionar datos. Varios procesos de gestión de datos están basados en CRUD, en los que **dichas operaciones están específicamente adaptadas a los requisitos del sistema y de usuario**, ya sea para la gestión de bases de datos o para el uso de aplicaciones. Para los expertos, las operaciones son las herramientas de acceso típicas e indispensables para comprobar, por ejemplo, los problemas de la base de datos, mientras que para los usuarios, CRUD significa crear una cuenta (créate) y utilizarla (read), actualizarla (update) o borrarla (delete) en cualquier momento. Dependiendo de la configuración regional, las operaciones CRUD pueden implementarse de diferentes maneras, como lo muestra la siguiente tabla:

|  |  |  |  |
| --- | --- | --- | --- |
| **CRUD-Operation** | **SQL** | **RESTful HTTP** | **XQuery** |
| **Create** | INSERT | POST, PUT | insert |
| **Read** | SELECT | GET, HEAD | copy/modify/return |
| **Update** | UPDATE | PUT, PATCH | replace, rename |
| **Delete** | DELETE | DELETE | delete |

Frameworks CRUD: capa de acceso a las bases de datos

Si los objetos individuales son **visualizados por medio de una interfaz gráfica** y modificados con las llamadas operaciones CRUD, entonces se habla de un framework CRUD o de un CRUD grid. Por lo general, se trata de **interfaces HTML**. Un framework CRUD demanda varios pasos de transacción, de forma que los datos no se recogen una vez se han introducido, sino que es necesario pulsar la opción “Guardar” o “Continuar”. Las operaciones de un framework CRUD pueden aplazarse para ser ejecutadas en diferentes plazos, sin que los datos de dichos periodos de tiempo se bloqueen para otros usuarios. Este hecho resulta de gran importancia para **sistemas multiusuario**, pues permite que varias personas lean los mismos datos al mismo tiempo.

Para llevar a cabo las operaciones se utilizan las denominadas capas de persistencia, que, por lo general, pueden ser implementadas o están contenidas en forma de extensiones (módulos) en el framework. Estas rompen con la representación relacional y tabular de la totalidad de los datos para, en su lugar, **presentarlos en un nivel orientado a objetos**. Los frameworks CRUD facilitan el acceso al sistema de bases de datos y son utilizados tanto en el desarrollo como en el uso de aplicaciones. Existen numerosos frameworks con un concepto CRUD basados en diferentes lenguajes y plataformas. A continuación, presentamos algunos ejemplos:

|  |  |
| --- | --- |
| **Lenguaje o plataforma** | **Framework** |
| Java | JDBC (The Java Database Connectivity), Hibernate, JBoss Seam, Isis |
| PHP | Yii, CakePHP, Zikula, Symfony, TYPO3 Flow |
| Perl | Catalyst, Gantry |
| Python | Django, SQLAlchemy, web2py |
| Groovy | Grails |
| .NET | NHibernate, ADO.NET/Entity Framework |
| Ruby | Ruby on Rails |
| JavaScript | Backbone.js, AngularJS |

# REST API

El término REST (Representational State Transfer) se originó en el año 2000, descrito en la tesis de Roy Fielding, padre de la especificación HTTP. Un servicio REST no es una arquitectura software, sino un conjunto de restricciones con las que podemos crear un estilo de arquitectura software, la cual podremos usar para crear aplicaciones web respetando HTTP.  
  
Hoy en día la mayoría de las empresas utilizan API REST para crear servicios. Esto se debe a que es un estándar lógico y eficiente para la creación de servicios web. Por poner algún ejemplo tenemos los sistemas de identificación de Facebook, la autenticación en los servicios de Google (hojas de cálculo, Google Analytics, …).

Según Fielding las restricciones que definen a un sistema RESTful serían:

* **Cliente-servidor**: esta restricción mantiene al cliente y al servidor débilmente acoplados. Esto quiere decir que el cliente no necesita conocer los detalles de implementación del servidor  y el servidor se “despreocupa” de cómo son usados los datos que envía al cliente.
* **Sin estado**: aquí decimos que cada petición que recibe el servidor debería ser independiente, es decir, no es necesario mantener sesiones.
* **Cacheable**: debe admitir un sistema de almacenamiento en caché. La infraestructura de red debe soportar una caché de varios niveles. Este almacenamiento evitará repetir varias conexiones entre el servidor y el cliente para recuperar un mismo recurso.
* **Interfaz uniforme**: define una interfaz genérica para administrar cada interacción que se produzca entre el cliente y el servidor de manera uniforme, lo cual simplifica y separa la arquitectura. Esta restricción indica que cada recurso del servicio REST debe tener una única dirección, “URI”.
* **Sistema de capas**: el servidor puede disponer de varias capas para su implementación. Esto ayuda a mejorar la escalabilidad, el rendimiento y la seguridad.

Algunas características de una API REST

-Las operaciones más importantes que nos permitirán manipular los recursos son cuatro: GET para consultar y leer, POST para crear, PUT para editar y DELETE para eliminar.

-El uso de hipermedios (término que en el ámbito de las páginas web define el conjunto de procedimientos para crear contenidos que contengan texto, imagen, vídeo, audio y otros métodos de información) para permitir al usuario navegar por los distintos recursos de una API REST a través de enlaces HTML.

Para terminar, comentar que lo más importante a tener en cuenta al crear nuestro servicio o API REST no es el lenguaje en el que se implemente sino que las respuestas a las peticiones se hagan en XML o JSON, ya que es el lenguaje de intercambio de información más usado.

# MONGODB Gestor De Base De Datos Nosql.

Las bases de datos convencionales son de **tipo relacional**, es decir, constan de **listas tabulares de datos (también conocidas como "relaciones de datos"**, de ahí el verdadero origen de su nombre), que se relacionan entre sí para almacenar de la forma más eficiente posible la información. Se consultan [usando SQL](https://www.campusmvp.es/recursos/post/Que-es-el-lenguaje-SQL.aspx) (*Structured Query Language*), un lenguaje estructurado para gestionar la información.

En los últimos años, Internet ha crecido de forma espectacular dando lugar a nuevos tipos de aplicaciones que necesitan gestionar cantidades enormes de datos, que deben ser introducidos y extraídos rápidamente y que deben poder escalar/crecer de manera sencilla y barata. En estos sistemas **importa más la flexibilidad, la velocidad y la capacidad de escalado horizontal** que otras cuestiones tradicionalmente cruciales como la consistencia o disponer de una estructura perfectamente definida para los datos. Es más, las tradicionales [características ACID](https://es.wikipedia.org/wiki/ACID) de las bases de datos relacionales no son una premisa para los nuevos gestores de información.

Por todo ello existe una tendencia imparable hacia la definición y uso de un nuevo tipo de **bases de datos de tipo documental**, que en lugar de registrar relaciones de datos almacenan **documentos con estructuras arbitrarias** y cumplen con las premisas expuestas anteriormente. Sus **principales características** definitorias son, por un lado, la **enorme flexibilidad** que brindan en los esquemas de datos a la hora de almacenar la información y su simplicidad de uso, pero también su **altísimo rendimiento** y **escalabilidad**, la **facilidad de mantenimiento** y la capacidad de funcionar **sin puntos de fallo** pues pueden recuperarse aunque se caiga cualquiera de sus nodos.

Además, la forma de consultarlas y extraer y resumir información es completamente distinta a lo que estamos acostumbrados con SQL, por eso a este tipo de datos documentales se les denomina también **bases de datos no-relacionales**, documentales o más comúnmente **Bases de datos NoSQL**(de *Not Only SQL*).

**OJO**: Se suele confundir el concepto NoSQL con el de **base de datos documental**. En realidad éstas últimas son una categoría concreta dentro de las bases de datos NoSQL que también incluyen **almacenes de pares clave-valor** (como Dynamo o MotionDb), servidores de **estructuras de datos** (como Redis), **almacenes de N-Tuplas** (como Apache River) o **almacenes de filas** (como Cassandra o BigTable, desarrollada por Google), entre otros tipos. Conviene no confundirlas.

**Escalabilidad**

La escalabilidad de un sistema informático se define como **la capacidad de un sistema para expandirse** según las necesidades que se deriven del uso que se le va a dar. De este modo, por ejemplo, una aplicación web es escalable si es posible atender a un número cada vez mayor de usuarios sin necesidad de cambiar la aplicación. Fundamentalmente existen **dos maneras de escalar**:

* **Escalado vertical**: implica añadir más recursos al sistema actual para que pueda atender más solicitudes. Así se le añaden más procesadores o memoria a un servidor o se añade más espacio de almacenamiento. El escalado vertical tiene un límite claro y llega un punto en el que la aplicación necesita ser rediseñada para poder escalar más. En inglés a este tipo de escalado se le denomina "Scale Up".
* **Escalado horizontal**: en este caso el escalado se consigue simplemente añadiendo más nodos al sistema, por ejemplo, poniendo otra máquina en paralelo a funcionar. Este tipo de escalado es más sencillo y no tiene límite, pero implica que las aplicaciones estén adaptadas para poder hacerlo, pues implican coordinación, replicación, etc... Se denomina comúnmente como "Scale Out".

Los sistemas NoSQL son sistemas de almacenamiento de información **ágiles**, de **gran rendimiento**, **distribuidos** y, en casi todos los casos, **escalables horizontalmente**.

MongoDB

[**MongoDB**](https://www.mongodb.org/) es un gestor de datos **NoSQL distribuido de tipo documental** que almacena documentos en un formato similar a [JSON](https://es.wikipedia.org/wiki/JSON) (para ser más exactos internamente usa BSON). Está escrita en C++ y es **multi-plataforma**, **Open Source y gratuito**.

El proyecto nació a finales del año 2007 como un proyecto interno de una empresa llamada **10Gen** para usarlo en una aplicación de Internet que estaban desarrollando, pero en 2009 decidieron liberarlo como Open Source y dedicarse íntegramente a él, ofreciendo soporte comercial y servicios relacionados.

Su nombre proviene de la palabra en inglés **Humongous**, que significa literalmente "algo realmente grande", y se refiere a su capacidad de gestionar cantidades enormes de datos.

Sus principales características son:

* Basado en el motor V8 de Google Chrome para JavaScript. Facilidad de aprendizaje por basarse en este lenguaje.
* Almacenamiento flexible basado en JSON sin necesidad de definir esquemas previamente.
* Soporte para creación de índices a partir de cualquier atributo, lo que facilita mucho su uso para porque no es necesario definir procesos [Map-Reduce](https://es.wikipedia.org/wiki/MapReduce" \t "_blank).
* Alto rendimiento para consultas y actualizaciones.
* Consultas flexibles basadas en documentos.
* Alta capacidad de crecimiento, replicación y escalabilidad: puedes escalar horizontalmente simplemente añadiendo máquinas baratas sin ver afectado el rendimiento ni complicar la gestión.
* Soporte para almacenamiento independiente de archivos de cualquier tamaño basado en GridFS.
* Soporte para tareas Map-Reduce si es necesario.

**Aplicaciones de MongoDB**

Este tipo de bases de datos no sustituyen a las bases de datos tradicionales, como SQL Server, Oracle o MySQL, sino que **las complementan para ciertos tipos de aplicaciones** especializadas. Las bases de datos documentales como MongoDB se utilizan para multitud de tareas, pero fundamentalmente cuando necesitamos **flexibilidad**en la definición de los datos, **sencillez**a la hora de acceder a éstos, **gran rendimiento** y posibilidad de **crecer muy rápido**.

Es adecuada para crear **aplicaciones de Internet** que registren muchos datos o que simplemente queramos crear de manera muy flexible, pero también para **sistemas muy grandes** como registradores de datos de sensores, que pueden llegar a recibir decenas o cientos de miles de lecturas de datos por segundo, pasando por **gestores de datos de ventas**, infraestructura de almacenamiento para **redes sociales**, **juegos masivos online**, gestores de **contenidos**, aplicaciones de **análisis de datos** y *reporting...*

Actualmente lo utilizan para empresas como eBay, Foursquare, SourceForge, The New York Times, The Guardian, SAP, o el propio CERN en su gran colisionador de hadrones, pero también muchas empresas pequeñas que quieren poder desarrollar de manera ágil, barata, sencilla y sin miedo a poder crecer más adelante.

Al estar basada en JavaScript se lleva especialmente bien con todo tipo de aplicaciones web, especialmente las más modernas *Single Page Applications*.

En general **MongoDB puede utilizarse para casi cualquier cosa para la que utilizarías SQL Server o MySQL**, pero sin la rigidez que presentan este tipo de bases de datos.

En [este interesante enlace](http://highscalability.com/blog/2010/12/6/what-the-heck-are-you-actually-using-nosql-for.html) (en inglés) encontrarás una lista exhaustiva de aplicaciones típicas de las bases de datos NoSQL en general.

A lo largo de las próximas semanas -fundamentalmente a través de vídeos- vamos a ir viendo los fundamentos de trabajo con MongoDB: su instalación en varios sistemas operativos, sus conceptos básicos importantes y uso básico desde algún lenguaje, como C#.