Diplomatura en Django:

**Curso de Django Nivel Avanzado – En producción**

Módulo 2:

**Configuraciones finales**

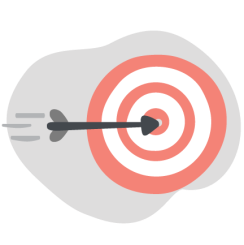
Unidad 3:

**SEO**

# 

### Presentación

En esta unidad trabajaremos sobre los aspectos principales del posicionamiento natural del sitio web.



### Objetivos

**Que los participantes logren…**

Comprender cuales son los aspectos fundamentales en lo que respecta al SEO.

# 

### Bloques temáticos

1. Funcionalidades de búsqueda.

2. Incorporación de Microdatos.

3. Utilización de Json-ld.

4. Optimizar La búsqueda de nuestro negocio.

5. Tipo de funcionalidades.

6. XML.

7. DTD.

8. XML Schema - Sitemap.xml.

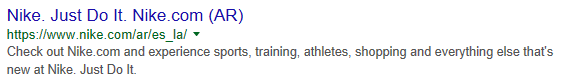
# 1. Funcionalidades de búsqueda.

Si diseñamos nuestro sitio de forma correcta y proporcionamos información extra sobre el contenido, podemos modificar como el motor de búsqueda ve, comprende y muestra nuestras páginas, habilitando funcionalidades extras que pueden enmarcarse dos categorías.

* **Tipo de contenido**: Muchas características de búsqueda están relacionadas con el tema de la página. Por ejemplo, si la página tiene una receta o un artículo de noticias, o contiene información sobre un evento o un libro, los resultados de búsqueda de Google pueden aplicar características específicas del contenido, como hacer que la página sea elegible para aparecer en un carrusel de noticias, un carrusel de recetas o una lista de eventos.
* **Mejoras**: Estas son características que se pueden aplicar a más de un tipo de contenido. Por ejemplo, proporcionar estrellas de revisión para una receta o película, o exponer un carrusel de resultados enriquecidos.

No hay garantía de que nuestra página aparezca en los resultados de búsqueda con la función especificada. Esto se debe a que las funciones de búsqueda dependen de muchos factores, incluidos el tipo de dispositivo de búsqueda, la ubicación y si Google cree que la característica proporcionará la mejor experiencia de búsqueda para el usuario.

El sistema de búsqueda de Google tiene algunos tipos generales de categorías de resultados de búsqueda. La mayoría de las características se aplican a categorías específicas, por ejemplo, tomemos la categoría de resultados básicos (Basic result) está presenta un determinado sitio con una url que lleva al link establecido.



Una mejora que se podría introducir en este tipo de categoría es presentar en lugar del link, un árbol de navegación del sitio, como podría ser:

Nike > Categoría-de-productos > productos-con-descuento

De esta forma podríamos guiar al usuario a la página que más se adapte al tipo de búsqueda que está realizando.

**¿Cómo agrego funcionalidades a mi sitio?**

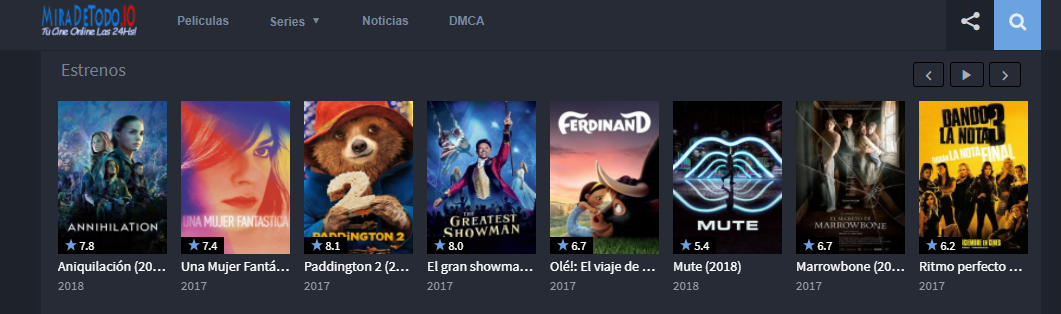
Las funcionalidades son introducidas mediante datos estructurados que pueden ser incorporados de diferentes maneras a nuestro sitio, las dos formas principales son mediante:

* Utilización de micro datos
* Utilización de plugin de javaScript desarrollado por google (json-ld). Este es el método recomendado por google.

# 2. Incorporación de Microdatos.

Los microdatos son datos extras introducidos dentro de las etiquetas html de nuestra maqueta destinados a realizar un detalle del contenido de cada etiqueta.

Para comprender como utilizarlos, supongamos que estamos realizando una página para presentar películas, y que tenemos una sección en donde presentamos las películas destacadas, seguramente tenemos una imagen, y una descripción de cada película agregada, como se puede ver a continuación.



El código podría estar presentado dentro de una lista de navegación de la siguiente manera:

|  |
| --- |
| <ol>  <li ><h3>Título 1</h3><p>Descripción 1</p></li>  <li><h3>Título 2</h3><p>Descripción 2</p></li>  <li><h3>Título 3</h3><p>Descripción 3</p></li>  <li><h3>Título 4</h3><p>Descripción 4</p></li>  <li><h3>Título 5</h3><p>Descripción 5</p></li>  </ol> |

Esto puede ser correcto para el usuario, pero no para el explorador de google, ya que a partir del código anterior, no puede comprender el tipo de contenido que estamos incluyendo en cada etiqueta.

Si modificamos un poco el código y lo dejamos como se muestra a continuación todo cambia.

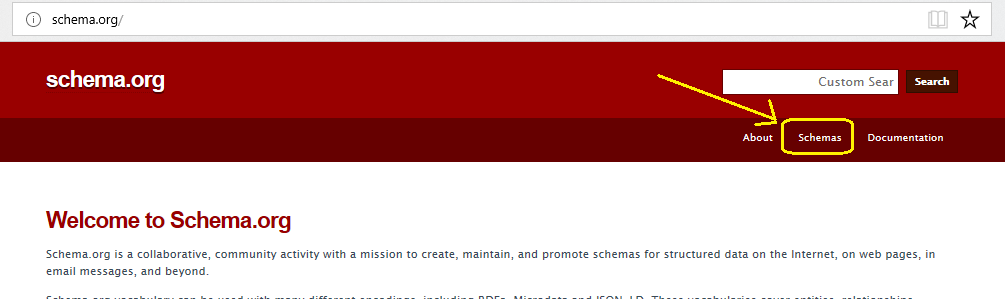
|  |
| --- |
| <ol itemscope itemtype="http://schema.org/Movie">  <li><h3 itemprop="name">Título 1</h3><p itemprop="genre">Descripción 1</p></li>  <li><h3>Título 2</h3><p>Descripción 2</p></li>  <li><h3>Título 3</h3><p>Descripción 3</p></li>  <li><h3>Título 4</h3><p>Descripción 4</p></li>  <li><h3>Título 5</h3><p>Descripción 5</p></li>  </ol> |

Aquí estamos utilizando microdatos mediante la incorporación de tres atributos:

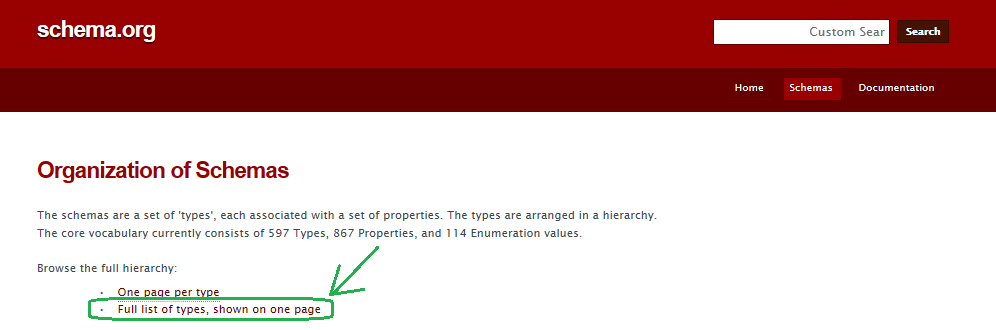
* itemscope:
* itemtype:
* itemprop:

Incluyendo “**itemscope**” especificamos que el HTML adentro de <ol>...</ol> es de un tipo específico, y estamos especificando el tipo de contenido con el atributo “**itemtype**” y la descripción de cada campo utilizando “**itemprop**”. En otras palabras le estamos indicando a google que se dirija a la página “http://schema.org/Movie” en donde encontrará un documento del tipo xml en el cual se encuentran detallados los componentes del tipo de documento específico que google asociara al contenido de la etiqueta que lo posee.

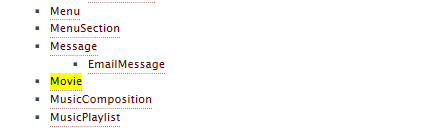
El documento xml puede ser creado por nosotros o podemos utilizar algún esquema que se encuentre en la web si tenemos el conocimiento necesario. Podemos encontrar muchos schemas predefinidos llendo a <http://schema.org/> y seleccionando del menú principal la opción “Schemas”



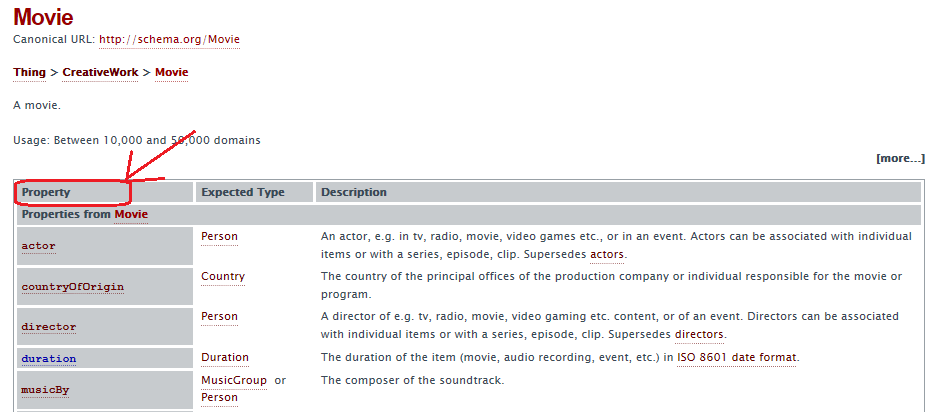
Una vez en la página nos dirigimos a la lista completa de opciones para buscar la categoría que mejor se adapte a nuestras necesidades.



El listado que podemos encontrar es muy grande y crece con el tiempo y el aporte de muchos desarrolladores. Si ahora buscamos y presionamos sobre la opción “Movie”



Nos encontraremos en una página en la cual podemos seleccionar la propiedad (nombre a colocar dentro de itemprop) junto a la descripción de lo que google interpreta al leer el campo que lo contiene.



Implementar metadatos de esta forma nos ayuda a explicarle a google de que trata nuestro sitio de manera que él decida a qué tipo de público le podría interesar nuestro contenido.

# 3. Utilización de Json-ld

La forma recomendada por google de incorporar datos estructurados es mediante la incrustación dentro de la cabecera o cuerpo de nuestro sitio de un plugin de javaScript. Tomemos como ejemplo un formulario de contacto.

|  |
| --- |
| <form id="formulario\_contacto" action=" " method="post">  <label for="nombre">Nombre</label>  <input id="nombre" type="text" size="50" maxlength="10" name="nombre"  placeholder="escribe tu nombre">  <label for="email">Email</label>  <input id ="email" type="text" size="50" maxlength="10" name="email" placeholder="escribe tu  email">  <label for = "comentario">Comentario</label>  <textarea id="comentario" name="comentario" rows="10" cols="40" placeholder="Agrega un  comentario"></textarea>  <input type="submit" value="Enviar ">  </form> |

A diferencia de los metadatos el código no está intercalado con el documento html facilitando su análisis y la utilización de herramientas adicionales. La forma de declarar el contenido de la página es adicionando un script como se muestra a continuación.

|  |
| --- |
| <script type="application/ld+json">  {  "@context": "http://schema.org",  "@type": "Organization",  "url": "http://adistancia.frh.utn.edu.ar/",  "name": "LIBROSONLINE.",  "contactPoint": {  "@type": "ContactPoint",  "telephone": " + 54 11 4443-7466",  "contactType": "Customer service"  }  }  </script> |

Aquí podemos notar varios puntos:

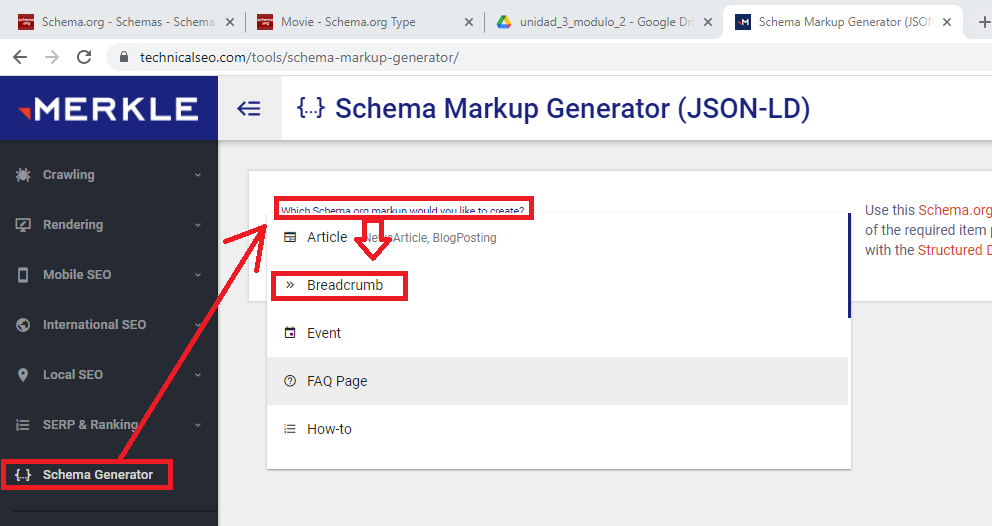
* La etiqueta de apertura de script, indica en su atributo type que estamos utilizando un script de descripción de sitio.
* El formato del script es de tipo “Json” como está indicado en el atributo type, el cual tiene el formato { clave1: valor1, clave2: valor2, ………claveN: valorN}, en donde las claves en este caso serían las propiedades incorporadas en los metadatos mediante itemprop y el valor el contenido de la etiqueta.
* La clave “@content” junto con la clave “@type” serían juntas como utilizar con microdatos: itemscope itemtype="http://schema.org/Organization "
* Cada clave puede a su vez contener un json con opciones.

## 3.1. Herramientas

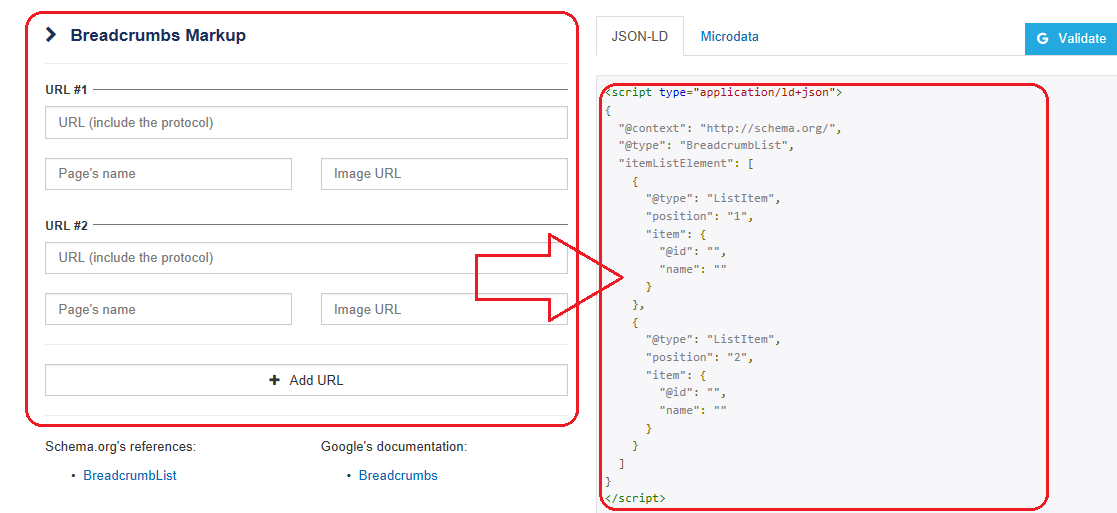
En la web podemos encontrar herramientas muy útiles para visualizar la estructura según lo ve el buscador y encontrar posibles errores en el código. Veamos algunas.

### 3.1.1. Generador de json-ld

Existen muchos generadores de código los cuales son muy útiles sobre todo cuando no tenemos mucho conocimiento de javaSript, es cuestión de buscar durante unos instantes para que nos salgan muchas opciones, por ejemplo podemos encontrar un generador que contiene la opción de arboles de navegación para la categoría de resultados más común, en: <https://technicalseo.com/seo-tools/schema-markup-generator/>

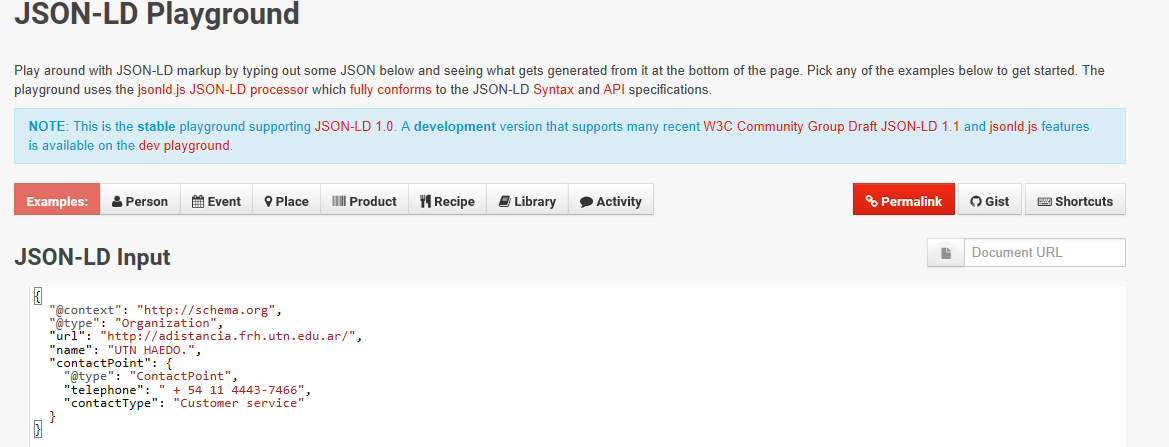


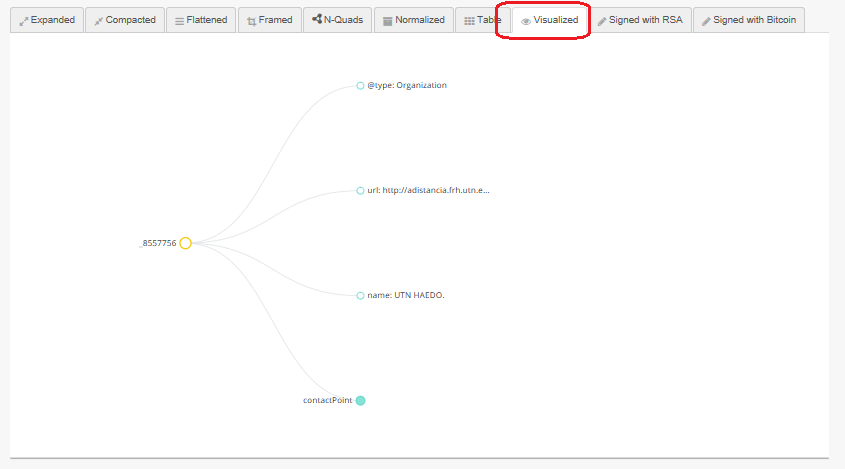
Si seleccionamos la opción “{…}Schema Generator > Which Schema org markup would you like to create > Breadcrumbs” del desplegable podemos a partir de un formulario, crear la estructura de navegación de forma rápida y sin conocimientos previos, lo único que debemos hacer es llenar los campos y copiar y pegar en nuestra página el código que nos retorna.



### 3.1.2. Visualizador de json-ld

Es muy interesante ver como el navegador interpreta nuestros datos de forma visual y como estructurar correctamente el código creado. Si accedemos a <https://json-ld.org/playground/> podemos pegar nuestro código y seleccionar la opción de visualización:

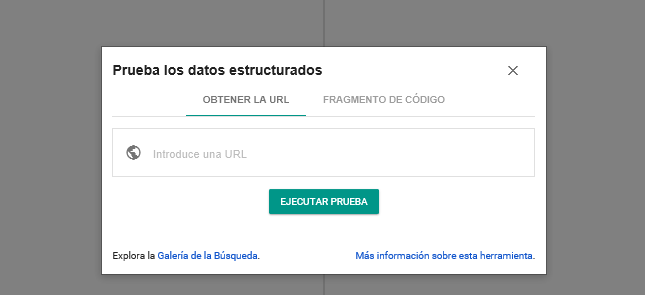




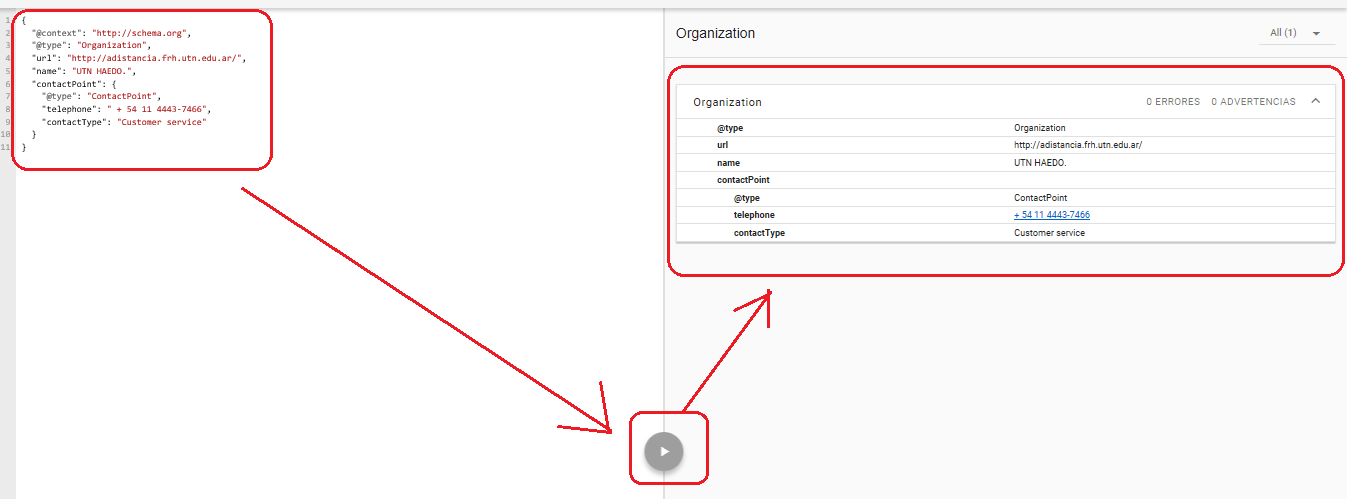
Como podemos comprobar, nuestra página es comprendida por el explorador como si fuera un árbol de nodos.

### 3.1.3. Testeo de datos

Finalmente luego de trabajar con el código que mejor se adapte a nuestro sitio, podemos testear el funcionamiento accediendo a: <https://search.google.com/structured-data/testing-tool/u/0/>



Si tenemos nuestra página web con su código ya establecido, podemos simplemente pegar la url del sitio, en caso contrario podemos pegar directamente el código seleccionando la opción “Fragmento de código”. En la pantalla que aparece a continuación debemos pegar el código en el campo de la izquierda y presionar el botón verde central para que se realice el test.



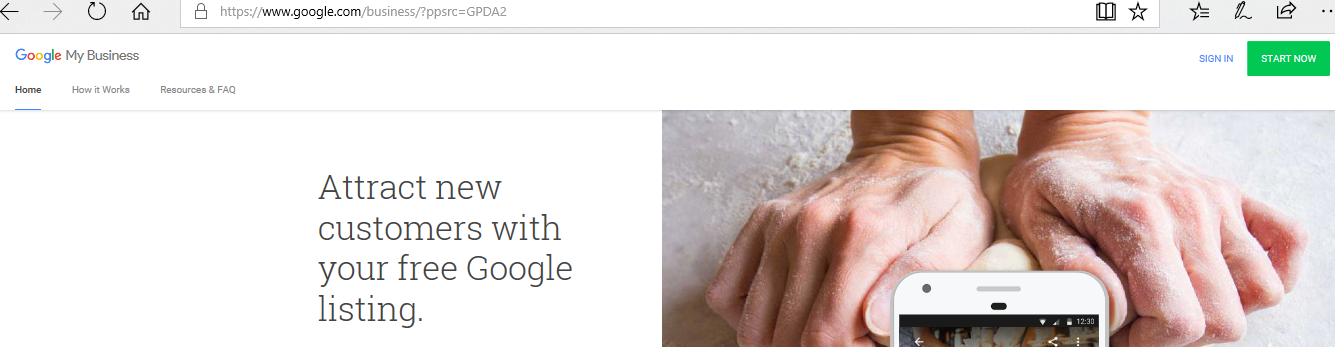
# 4. Optimizar La búsqueda de nuestro negocio.

Al agregar el marcado de datos estructurados a su sitio, se puede habilitar que aparezcan más elementos funcionales y visuales del sitio directamente en los resultados y en las tarjetas de graficas de presentación. Esto hace que sea más fácil para los usuarios reconocer un sitio oficial y acceder a la información que proporciona más fácilmente cuando realizan búsquedas.

Google ofrece una variedad de formas de proporcionar detalles comerciales claves de forma de ser mostrados a los usuarios en los resultados de búsqueda.

## Agregar un negocio local

Para reclamar un negocio físico existente o crear uno nuevo, se puede utilizar Google My Business: <https://www.google.com/business/?ppsrc=GPDA2>



Esto va a hacer que nuestro negocio sea mostrado a los clientes que se encuentran cerca de la zona en la cual nos encontramos, solo debes contar con un mail de gmail y ser el propietario o estar habilitado por el propietario del negocio, para que una vez verificado que somos propietarios, podamos proporcionar y editar la dirección, información de contacto, tipo de negocio y fotos. Esto permite que la información de nuestra empresa local aparezca en Google Maps y en las tarjetas de Gráfico de conocimiento.

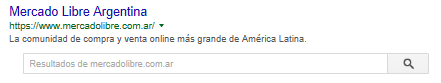
Los pasos son muy simples la página te va guiando de forma intuitiva para incorporar todos los datos necesarios.

# 5. Tipo de funcionalidades.

Además de los arboles de búsqueda (Breadcrumb), contamos con otros tipos de funcionalidades que.

## 5.1. Vínculos a sitios de búsqueda (Sitelinks searchbox)

La Búsqueda de Google puede exponer un cuadro de búsqueda con un alcance en el sitio web cuando aparece como un resultado de búsqueda. Este cuadro de búsqueda funciona con la Búsqueda de Google. El cuadro de búsqueda implementa sugerencias en tiempo real y otras características.



Un ejemplo podría ser:

|  |
| --- |
| <script type="application/ld+json">  {  "@context": "http://schema.org",  "@type": "WebSite",  "url": "https://www.example.com/",  "potentialAction": {  "@type": "SearchAction",  "target": "https://query.example.com/search?q={search\_term\_string}",  "query-input": "required name=search\_term\_string"  }  }  </script> |

Aquí debemos sustituir nuestra url por la que aparece de ejemplo.

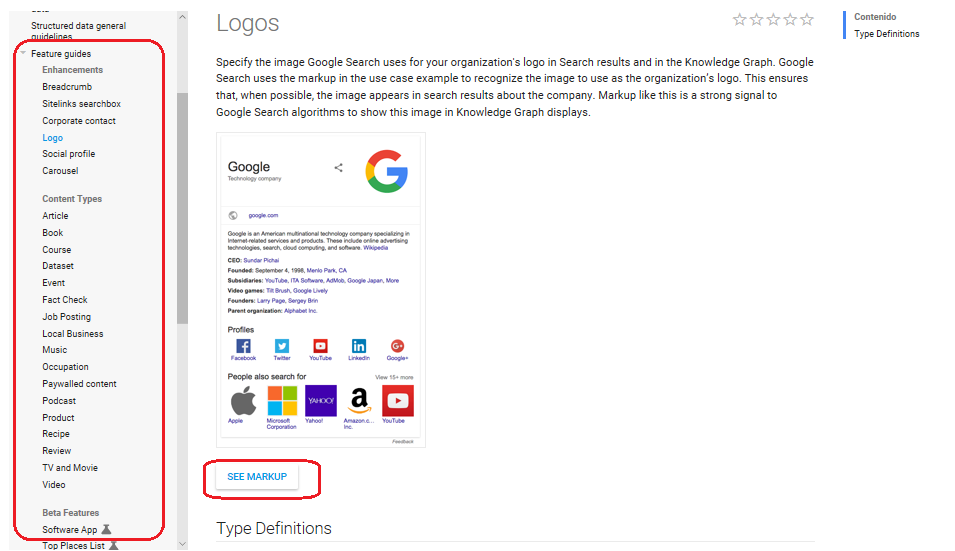
## 5.2. Logo

Especifique la imagen que el buscador de google utiliza para el logotipo de nuestra organización en los resultados de Búsqueda y en el Gráfico de conocimiento. Un ejemplo de código podría ser el siguiente, recordando sustituir los datos de ejemplo por los de nuestro negocio.

|  |
| --- |
| {  "@context": "http://schema.org",  "@type": "Organization",  "url": "http://www.example.com",  "logo": "http://www.example.com/images/logo.png"  } |

**Ok pero ¿Dónde puedo encontrar una referencia a los tipos de funcionalidades y su forma de implementación?**

Ahora que tenemos una idea del funcionamiento podemos ingresar a la página de funcionalidades de google y encontrar en el menú lateral izquierdo cada una de las funcionalidades que podemos adicionar, y presionar en el botón “SEE MARKUP” de forma de ver un código de ejemplo que podemos utilizar con los datos de nuestro sitio.



# 6. XML

Los schemas son documentos xml, el cual es un lenguaje que fue desarrollado en principio para la transferencia de datos pero posteriormente también se utilizo para la creación de las vistas de aplicaciones como Android. XML (XML eXtensible Markup Language) es un lenguaje de etiquetas extensible, del cual se origina html el cual a diferencia de xml posee etiquetas preestablecidas.

## Diferencias entre XML Y HTML:

* A diferencia de HTML, en XML podemos ampliar el número de etiquetas, por lo cual decimos que XML es extensible.
* XML no dice como se presenta la información en pantalla en aplicaciones web, pero si nos permite estructurar y organizar grandes cantidades de información, la cual podemos almacenar y intercambiar posteriormente. Con estructurado se quiere decir, que permite indicar información sobre el contenido de las etiquetas
* Permite transferir información estructurada.
* Es un lenguaje de marcadores.
* Tiene la misma sintaxis para los comentarios que html.

Podemos encontrar una referencia al lenguaje en: <http://www.w3.org/TR/REC-xml/>

## Fundamentos

Las etiquetas utilizadas son analizadas sintácticamente, mientras que el texto de las etiquetas es leído como datos de carácter.

XML permite estructurar la información, pero no dice nada de cómo aparece en pantalla. Al inicio del contenido del archivo sin que exista un espacio previo, se debe colocar la etiqueta de declaración de versión y tipo de codificación utilizada:

|  |
| --- |
| <?xml version="1.0" encoding="utf-8"?> |

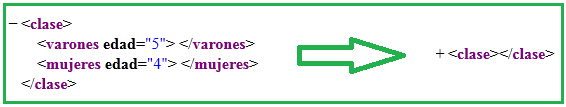
Un ejemplo podría ser:

|  |
| --- |
| xml1.xml |
| <?xml version="1.0" encoding="utf-8"?>  <!DOCTYPE MENSAJE SYSTEM "mensaje.dtd">  <clase>  <varones edad='5'>  </varones>  <mujeres edad='4'>  </mujeres>  </clase> |

En el ejemplo anterior podemos encontrar dos elementos muy importantes:

* La etiqueta raíz, en este caso <clase> que siempre debe de encontrarse presente en un documento xml. Dado que html se desprende de xml esta regla debe de cumplirse, en particular la etiqueta raíz en html se llama <html>.
* Al igual que en html no solo podemos declarar etiquetas, sino que dentro de las mismas podemos ubicar atributos.

Si intentamos abrir en un explorador el documento anterior nos aparece la estructura del documento, en donde cada etiqueta puede ser colapsada:



Un ejemplo un poco más completo podría ser:

|  |
| --- |
| xml2.xml |
| <?xml version="1.0" encoding="utf-8"?>  <!DOCTYPE MENSAJE SYSTEM "mensaje.dtd">  <mensaje>  <remite>  <nombre>Juan</nombre>  <email>juanbarretor@gmail.com</email>  </remite>  <destinatario>  <nombre>Pedro</nombre>  <email>pedrito@gmail.com</email>  </destinatario>  <asunto>Hola Pedrito</asunto>  <texto>  <parrafo>Este es mi primer contacto</parrafo>  </texto>  </mensaje> |

Al igual que en html no puede existir solapamiento de etiquetas, por lo cual esto no sería permitido:

<1><2></1></2>

Pero esto si:

<1></1><2></2>

La estructura podría presentarse como un diagrama de árbol, que es la forma en la cual el explorador interpreta el documento:

Recordar que el documento solo puede contener una etiqueta raíz, la cual en este ejemplo es <mensaje>

A diferencia de en html5 los valores que toman los atributos siempre deben estar entre comillas simples o dobles.

## Prologo

La etiqueta de apertura también puede contener el atributo “standalone”, Los documentos que se apoyan en una DTD externa para su validación no pueden considerarse autónomos y deben llevar: standalone=“no”

## Atributos

Los atributos pueden ser declarados de varias maneras. Ver ejemplo adjunto xml3.xml, pero no son recomendables de ser utilizados con cadenas de texto muy largas.

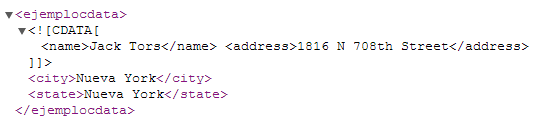
## Entidades predefinidas

En XML se puede definir entidades con caracteres especiales, para interpretar caracteres. Seguramente si has creado documento html antes lo has utilizado en tus páginas web mínimamente para adicionar un espacio en blanco.

|  |  |
| --- | --- |
| Entidad | Caracter |
| &amp; | & |
| &lt; | < |
| &gt; | > |
| &apos; | ‘ |
| &quot; | “ |

Existe otra construcción en XML que permite especificar datos, utilizando cualquier carácter, especial o no, sin que se interprete como marcado XML. La razón de esta construcción llamada CDATA(Character Data) es que a veces es necesario para los autores de documentos XML poder leerlo fácilmente sin tener que descifrar código de entidades. Es decir que los CDATA Son secciones de datos no analizados sintácticamente, se emplea en los documentos XML para bloquear texto que tiene que ser puesto a un lado por el analizador XML

|  |
| --- |
| xml4.xml |
| <?xml version="1.0" encoding="utf-8"?>  <ejemplocdata>  <![CDATA[  <name>Jack Tors</name>  <address>1816 N 708th Street</address>  ]]>  <city>Nueva York</city>  <state>Nueva York</state>  </ejemplocdata> |



En este caso los elementos “name” y “addres” no son reconocidos como marcadores XML. Notar que aparece como texto.

# 7. DTD

DTD (Document Type Definition) - Declaración del tipo de documento, es un documento que puede ser interno o externo al documento xml y nos permite definir qué tipo de datos contiene el documento así como validarlo.

La DTD define los tipos de elementos, atributos y entidades permitidas, y puede expresar algunas limitaciones para combinarlos.

Los documentos XML que se ajustan a su DTD, se denominan “validos”. El concepto de “validez” no tiene nada que ver con el de estar “bien-formado”. Un documento bien formado simplemente respeta la estructura y sintaxis definidas por la especificación de XML. Un documento bien formado puede además ser válido si cumple las reglas de una DTD determinada.

Los documentos XML sin una DTD asociada, no son “validos”, pero tampoco “inválidos” , simplemente son bien formados o no.

Los archivos DTD pueden ser externos o internos al documento.

Un DTD nos permite especificar:

1) Que elementos están permitidos.

2) Como se relacionan estos elementos.

3) Que tipo de elementos es cada uno.

4) Que nombre tiene cada elemento.

5) Los atributos requeridos y opcionales y si tienen valores por defecto.

## DTD interno

La declaración de tipo de documento comienza en la primera línea y termina con “]>”. Las declaraciones DTD son las líneas que empiezan con “<!ELEMENTO” y se denominan declaraciones de tipo elemento. También se pueden declarar atributos, entidades y anotaciones para una DTD.

Veamos un ejemplo:

|  |
| --- |
| xml5.xml |
| <?xml version="1.0" encoding="utf-8"?>  <!DOCTYPE etiqueta[  <!ELEMENT etiqueta (nombre, calle, ciudad)>  <!ELEMENT nombre (#PCDATA)>  <!ELEMENT calle (#PCDATA)>  <!ELEMENT ciudad (#PCDATA)>  ]>  <!--Esta parte es el XML -->  <etiqueta>  <nombre>Gaby</nombre>  <calle>Medrano 951</calle>  <ciudad>Bs. As. </ciudad>  </etiqueta> |

## DTD externo

Cuando trabajamos con una DTD externa al igual que sucede cuando nos llevamos el contenido de las etiquetas <style> de estilos en html o de incrustación de scripts <script> a un archivo aparte con extensión .css o .js, el código en el archivo externo no incluyen las etiquetas.

|  |
| --- |
| xml6.xml |
| <?xml version="1.0" encoding="utf-8“ standalone=“no”?>  <!DOCTYPE etiqueta SYSTEM “etiqueta.dtd">  <!--Esta parte es el XML -->  <etiqueta>  <nombre>Gaby</nombre>  <calle>Medrano 951</calle>  <ciudad>Bs. As. </ciudad>  </etiqueta> |

|  |
| --- |
| etiqueta.dtd |
| <!ELEMENT etiqueta (nombre, calle, ciudad)>  <!ELEMENT nombre (#PCDATA)>  <!ELEMENT calle (#PCDATA)>  <!ELEMENT ciudad (#PCDATA)> |

## Documentos validos.

Un documento válido debe cumplir:

* El documento debe estar bien construido.
* El nombre del elemento del root del documento debe coincidir con el nombre de la declaración de tipo de documento.

|  |
| --- |
| <!DOCTYPE etiqueta SYSTEM “etiqueta.dtd"> |

* El documento debe tener una DTD que declare todos los elementos, atributos y entidades que se utilicen en el documento.
* El documento debe adherirse a la gramática que establezca la DTD.

Nota: Todos los documentos válidos están bien construidos, pero no todos los documentos bien construidos son documentos válidos.

## Trabajar con elementos

Los elementos se declaran en la DTD con declaraciones de elementos, que adoptan la siguiente forma:

|  |
| --- |
| <!ELEMENT nombreElemento Type> |

Los tipos pueden ser:

1. **EMPTY.** El elemento no tiene contenido, pero puede contener atributos.

|  |
| --- |
| <!ELEMENT img EMPTY>  <img src = “Title.gif”/> |

1. **Sólo elementos.** Un elemento de sólo elemento no contiene más que elementos secundarios.

|  |
| --- |
| <!ELEMENT resume (intro, (education ¦ experience)+, hobbies?, references\*)> |

**¦** separa elemento de grupo de alternativas

Sin símbolo, el elemento secundario debe aparecer una sola vez

**?** El elemento secundario debe aparecer una sola vez o ninguna.

**\*** El elemento secundario puede aparecer cualquier número de veces.

**+** El elemento secundario debe aparecer por lo menos una vez.

1. **Mixto**. El elemento contiene una combinación de elementos secundarios y datos de caracteres **(#PCDATA)**. En estos casos siempre se coloca primero #PCDATA.

|  |
| --- |
| <!ELEMENT nombreElemento (#PCDATA ¦ title ¦ subject )> |

1. ANY. El elemento contiene cualquier contenido que permita la DTD

## Atributos

Dentro de un elemento, los atributos se usan para formar un par de nombre – valor, que en cierto modo describe una propiedad particular del elemento. La estructura es:

|  |
| --- |
| <!ATTLIST nombreElemento nombreAttr tipoAtt Default> |

El valor Default indica el uso, el cual puede ser:

|  |  |
| --- | --- |
| #REQUIRED | El atributo es obligatorio. |
| #IMPLIED | El atributo es opcional. |
| #FIXED value | El atributo tiene un valor fijo. |
| #default | El valor predeterminado del atributo. |

Los tipos pueden ser:

|  |  |
| --- | --- |
| Tipo | Descripción |
| CDATA | Datos de caracteres no analizados sintácticamente. Enumerados. Una serie de caracteres. |
| NOTATION | Una notación declarada en alguna otra parte de la DTD. |
| ENTITY | Una entidad binaria externa. |
| ENTITIES | Múltiples entidades binarias externas separadas por espacio en blanco. |
| ID | Un identificador único. |
| IDREF | Una referencia a un ID declarado en alguna otra parte de la DTD. |
| IDREFS | Múltiples referencias a los ID declarados en alguna otra parte del documento. |
| NMTOKEN | Un nombre compuesto por caracteres con símbolos XML(letras, números, puntos, guiones, dos puntos y subrayados) |
| NMTOKENS | Múltiples nombres compuestos por caracteres con símbolos XML. |

Veamos algunos ejemplos:

### Atributos de cadena

En este caso el equipo al que pertenece un jugador es un atributo de datos de caracteres **obligatorios** del elemento player.

En este caso el equipo al que pertenece un jugador es un atributo de datos de caracteres **opcional** del elemento player.

### Atributos enumerados

En este caso existe la obligación de seleccionar un valor de una lista al usar el atributo position. Si se opta por no definir el atributo, se presupondrá el valor predeterminado de “center”

### Declaración de Múltiples atributos.

Ver ejemplo integrador “001 Integrador.xml“

## Declaración de entidades

Las entidades son contenedores de almacenamiento de datos, y son los bloques de construcción con los que construyen los documentos XML, pueden ser:

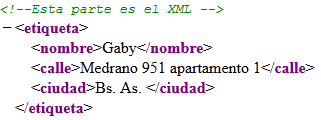
Por ejemplo las entidades generales son analizadas sintácticamente y están diseñadas para ser utilizadas en el contenido del documento.

No es posible hacer referencia a una entidad amenos que se haya declarado, y estas deberán ubicarse en la DTD

Veamos un ejemplo de entidades generales internas, en donde el nombre de la entidad se coloca entre los caracteres “&” y “;”

|  |
| --- |
| xml9.xml |
| <?xml version="1.0" encoding="utf-8"?>  <!DOCTYPE etiqueta[  <!ELEMENT etiqueta (nombre, calle, ciudad)>  <!ELEMENT nombre (#PCDATA)>  <!ELEMENT calle (#PCDATA)>  <!ELEMENT ciudad (#PCDATA)>  <!ENTITY reemplazar "apartamento 1">  ]>  *<!--Esta parte es el XML -->*  <etiqueta>  <nombre>Gaby</nombre>  <calle>Medrano 951 &reemplazar;</calle>  <ciudad>Bs. As. </ciudad>  </etiqueta> |

Al abrir la página en un explorador nos figura como:



# XML Schema - Sitemap.xml

Dentro de nuestro sitio web debemos colocar un archivo sitemap.xml que contenga la ruta a las páginas web de nuestro sitio con una fecha de modificación, un periodo de tiempo que es utilizado para decirle a google cada cuanto debe regresar a ver las modificaciones realizadas, y un valor de prioridad que se encuentra entre cero y uno.

El documento contiene una etiqueta raíz <urlset> y cada página va dentro de las etiquetas <url>, en este caso la validación no se realiza mediante una DTD sino mediante un XML Schema.

|  |
| --- |
| sitemap.xml |
| <?xml version="1.0" encoding="UTF-8"?>  <urlset xmlns="http://www.sitemaps.org/schemas/sitemap/0.9">  <url>  <loc>http://www.dominio.com/</loc>  <lastmod>2013-07-19</lastmod>  <changefreq>weekly </changefreq>  <priority>0.8</priority> </url>  …    </urlset> |

Al igual que las DTD, XML Schema se utiliza para validar los archivos xml. La estructura es más detallada que las DTD y permite una validación también más detallada.

* Las XML Schemas son creadas usando XML básico, en lugar de las DTD.
* Las XML Schemas soportan todas las recomendaciones sobre los Namespaces.
* Permiten validar elementos con contenido de texto y utilizar tipos de datos definidos.
* Permite crear más fácilmente modelos de contenidos complejos y reusables.
* Permite modelar conceptos de programación como herencia de objetos y tipo de sustitución.

La mayoría de los XML Schemas son guardados en un documento separado de forma similar a las DTD. El archivo XML contiene una referencia al XML Schema. Un documento xml que adhiere a un vocabulario particular de un XML Schema es denominado una instancia del XML Schema.

## Cabecera

La cabecera comienza con el elemento <schema>.

* Dentro se encuentra la declaración del namespace

xmlns=<http://www.w3.org/2001/XMLSchema>

* El atributo targetNamespace indica que estamos desarrollando un vocabulario para el Namespace: <http://www.example.com/name>
* No es necesario elegir el nombre target, podría ser cualquier otro.
* elementoFormatDefault, específica la forma en la cual el Namespace es utilizado dentro del documento XML.

|  |
| --- |
| <schema xmlns="http://www.w3.org/2001/XMLSchema"  xmlns:target="http://www.example.com/name"  targetNamespace="http://www.example.com/name" elementFormDefault="qualified"> |

Entrar en detalle sobre los xmls Schemas llevaría un curso completo, podemos presentar a continuación un ejemplo de implementación sin entrar en detalles.

|  |
| --- |
| name.xml |
| <?xml version="1.0"?>  <name  xmlns="http://www.example.com/name"  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  xsi:schemaLocation="http://www.example.com/name name.xsd"  title="Mr.">  <first>John</first>  <middle>Fitzgerald Johansen</middle>  <last>Doe</last> |

|  |
| --- |
| name.xsd |
| <?xml version="1.0"?>  <schema xmlns="http://www.w3.org/2001/XMLSchema"  xmlns:target="http://www.example.com/name"  targetNamespace="http://www.example.com/name" elementFormDefault="qualified">  <element name="name">  <complexType>  <sequence>  <element name="first" type="string"/>  <element name="middle" type="string"/>  <element name="last" type="string"/>  </sequence>  <attribute name="title" type="string"/>  </complexType>  </element>  </schema> |

# sitemap

## Paso 1

Agregamos a settings.py

INSTALLED\_APPS = [

'django.contrib.admin',

'django.contrib.auth',

'django.contrib.contenttypes',

'django.contrib.sessions',

'django.contrib.messages',

'django.contrib.staticfiles',

---------

'django.contrib.sitemaps',

]

## Paso 2

Nos debemos asegurar que en templas la opción de app\_dirs esta en true

TEMPLATES = [

{

-------------

'APP\_DIRS': True,

------------------

},

]

## PASO 3

Agrego en settings.py la variable variable:

SITE\_ID=1

Y en el panel admin modifico el valor de SITES que viene por defecto “example.com” con mi nombre de dominio, en este caso “127.0.0.1:8000”

## PASO 4

Creo un archivo sitemap.py en cualquier lugar y lo llamo desde urls.py principal, a modo de ejemplo lo agregamos en la app “tienda”

Ejemplo hojas estáticas

|  |
| --- |
| sitemap.py |
| from django.contrib import sitemaps  from django.urls import reverse  class TiendaViewSitemap(sitemaps.Sitemap):      priority = 0.9      changefreq = 'daily'      def items(self):          return ['tienda',]      def location(self, item):          return reverse(item) |

En el caso de una url dinámica como es el caso de cada procto debo agregar en en models.py el método

|  |
| --- |
| def get\_absolute\_url(self):          return f"/tienda/ver\_producto/{self.articulo}/" |

Y un sitemap.py en este caso quedaría dentro de la app de Producto así:

|  |
| --- |
| sitemap.py |
| from django.contrib.sitemaps import Sitemap  from productos.models import Producto  class ProductoSitemap(Sitemap):      changefreq = "weekly"      priority = 0.7      def items(self):          return Producto.objects.filter(estado='Publicado') |

## PASO 5

Ahora modifico el urls.py principal así:

|  |
| --- |
| urls.py |
| from django.contrib import admin  from django.urls import path, include  from django.conf import settings  from django.conf.urls.static import static  from django.contrib.sitemaps.views import sitemap  from tienda.sitemap import TiendaViewSitemap  from productos.sitemap import ProductoSitemap  sitemaps = {      'tienda': TiendaViewSitemap,      'producto':ProductoSitemap,    }  urlpatterns = [      path("", include("presentacion.urls")),      path('admin/', admin.site.urls),      path("carrito/", include("carro.urls")),      path("contacto", include("contacto.urls")),      path("pagos/", include("pagos.urls")),      path("panel/", include("usuario.urls")),      path("pedidos/", include("pedidos.urls")),      path("tienda/", include("tienda.urls")),      path("transporte/", include("transporte.urls")),      path('sitemap.xml', sitemap, {'sitemaps': sitemaps}, name='django.contrib.sitemaps.views.sitemap'),  ] + static(settings.MEDIA\_URL, document\_root=settings.MEDIA\_ROOT)  if settings.DEBUG:      import debug\_toolbar      urlpatterns += path('\_\_debug\_\_/', include(debug\_toolbar.urls)), |

## PASO 6

Ya puedo acceder al dominio/sitemap.xml y ver el resultado





### Bibliografía utilizada y sugerida

#### Libros y otros manuscritos:

Django. (2022). The web framework for perfectionists with deadlines. Django. Recuperado de https://www.djangoproject.com/.

Apache. (2022). The Apache Software Fundation. Apache. Recuperado de https://www.apache.org/.

Launch Modern Applications with NGINX. Recuperado de https://www.nginx.com/.

Hace más con tu dinero. Mercado Pago. Recuperado de https://www.mercadopago.com.ar/.

Derek DeJonghe - NGINX Cookbook, 2nd Edition - O'Reilly Media, Inc., Year: 2022.