



UT05: DESARROLLO DE COMPONENTES

ÍNDICE

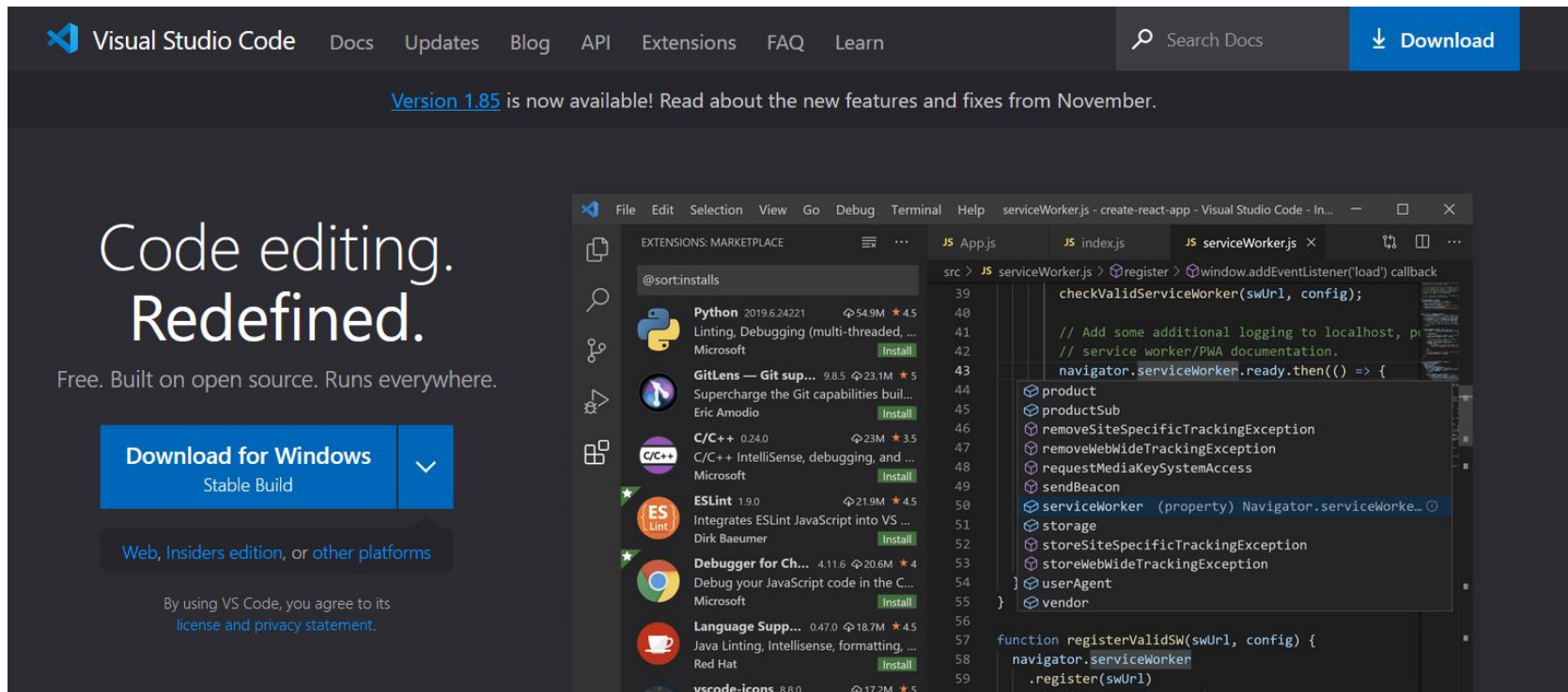
- 1.- Instalación de las herramientas de desarrollo
- 2.- Primer módulo con Odoo
- 3.- Estructura de un módulo en Odoo
- 4.- Modelo en Odoo
- 5.- Vistas en Odoo

1

INSTALACIÓN DE LAS HERRAMIENTAS DE DESARROLLO



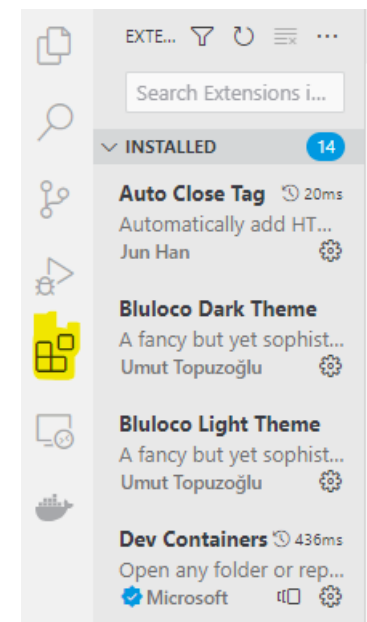
Para desarrollar en Odoo no es necesario ningún entorno en particular, pero personalmente recomiendo utilizar **Visual Studio Code**



VS Code tiene una gran adaptabilidad gracias a sus múltiples **extensiones**, por lo que algo importante es seleccionar las extensiones más adecuadas para el trabajo que vas a desarrollar.

A continuación, voy a indicar algunas de las que utilizo, pero simplemente es una sugerencia y puedes sentirte libre de utilizar las que mejor se adecúen a tu modo de trabajo

Para gestionar las extensiones tienes que ir al menú de la izquierda y seleccionar el icono con los cuatro cuadrados (o bien con el atajo ***Ctrl + Shift + X***)



Tema

El tema determina los colores que se utilizarán en general para toda la interfaz y en particular para el resaltado de sintaxis.

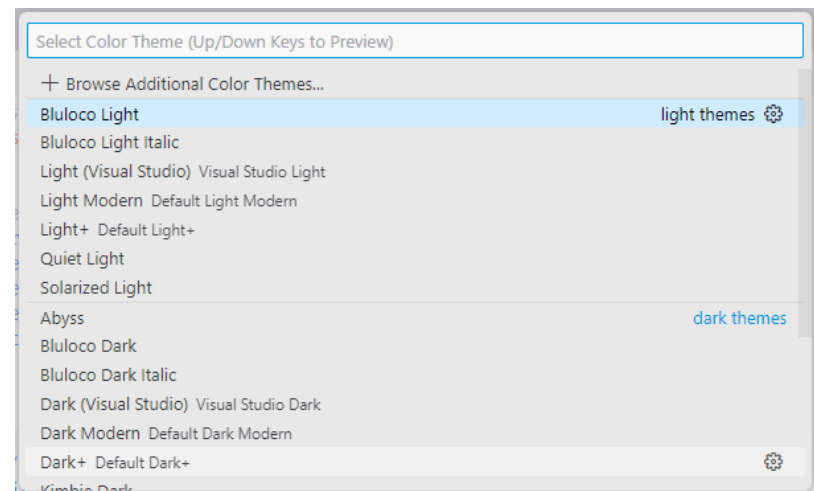
Hay múltiples temas y la elección de uno u otro es una cuestión muy personal, por lo que lo mejor es **probar unos cuantos** hasta encontrar uno con el que nos encontremos a gusto.

Para comenzar por algún sitio, en el enlace tienes algunas sugerencias de temas.

<https://hackr.io/blog/best-vscode-themes>



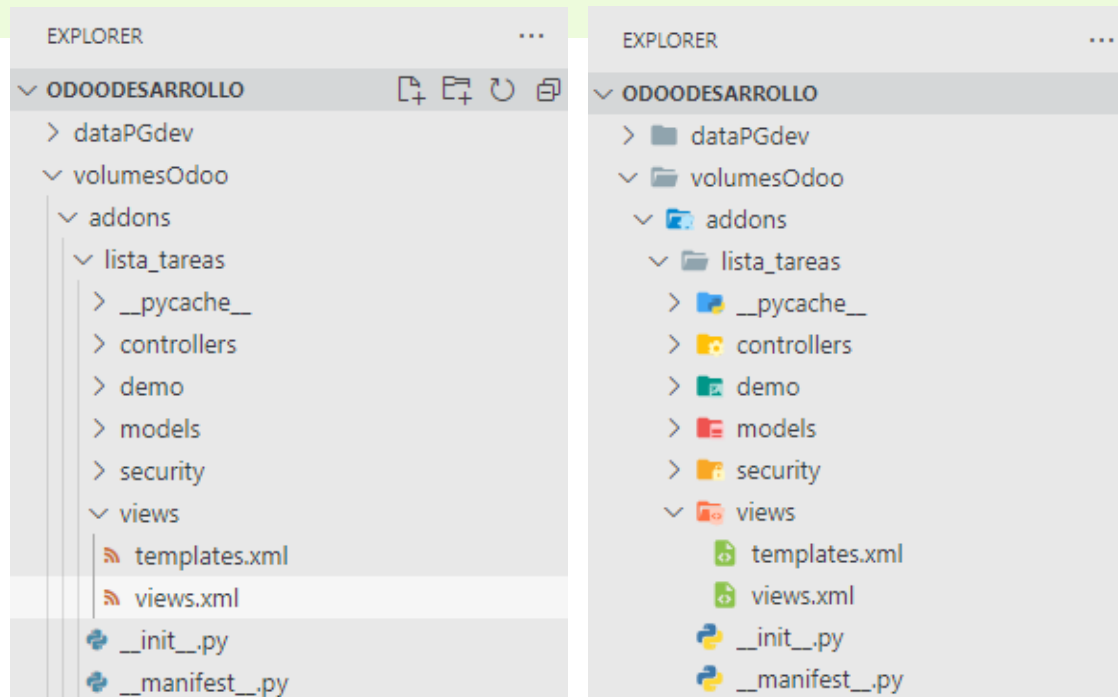
Para activar un tema una vez instalado pulsas **Ctrl+Mayus+P** y buscas **Preferences: Color theme** (en cuanto escribas *theme* te filtrará resultados)



Iconos

Por defecto, VS Code tiene una serie de iconos para identificar los tipos de archivos, pero hay extensiones que modifican estos iconos por otros más atractivos.

Personalmente, para este cometido, aconsejaría la instalación de **Material Icon Theme**. En las imágenes se ve la configuración por defecto (izq) y con Material (dcha).



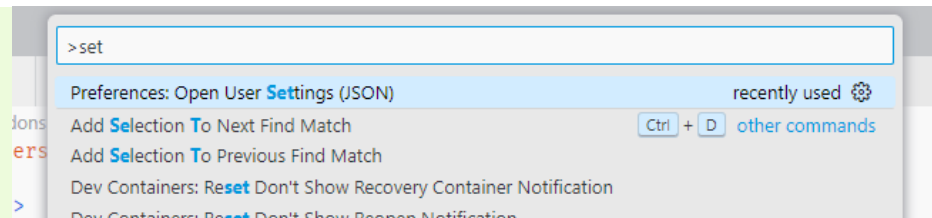
Fuente

La fuente también es algo importante, una opción muy clara y legible es **Fira Code**, que se puede descargar del enlace adjunto.

<https://github.com/tonsky/FiraCode>



Para activarla pulsamos **Ctrl+Mayus+P** y buscamos **Preferences: Open User Settings (JSON)**

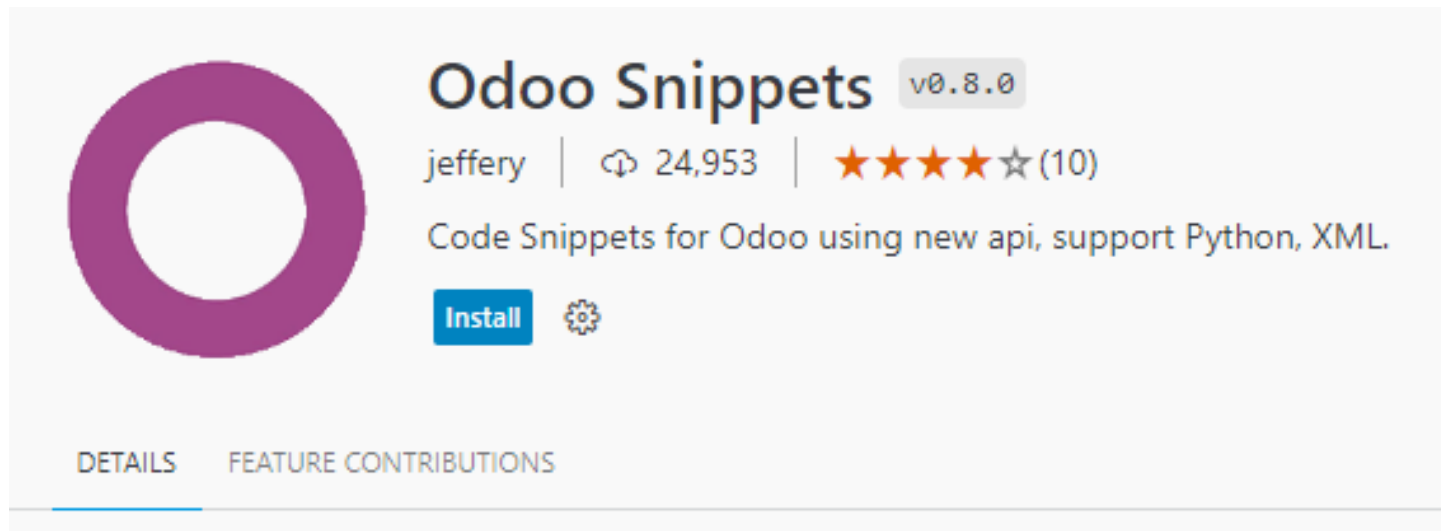


Y añadimos estas dos líneas en el archivo JSON que se abre.

```
"editor.fontFamily": "'Fira Code', Consolas, 'Courier New', monospace",  
"editor.fontLigatures": true,
```


Extensiones para Odoo

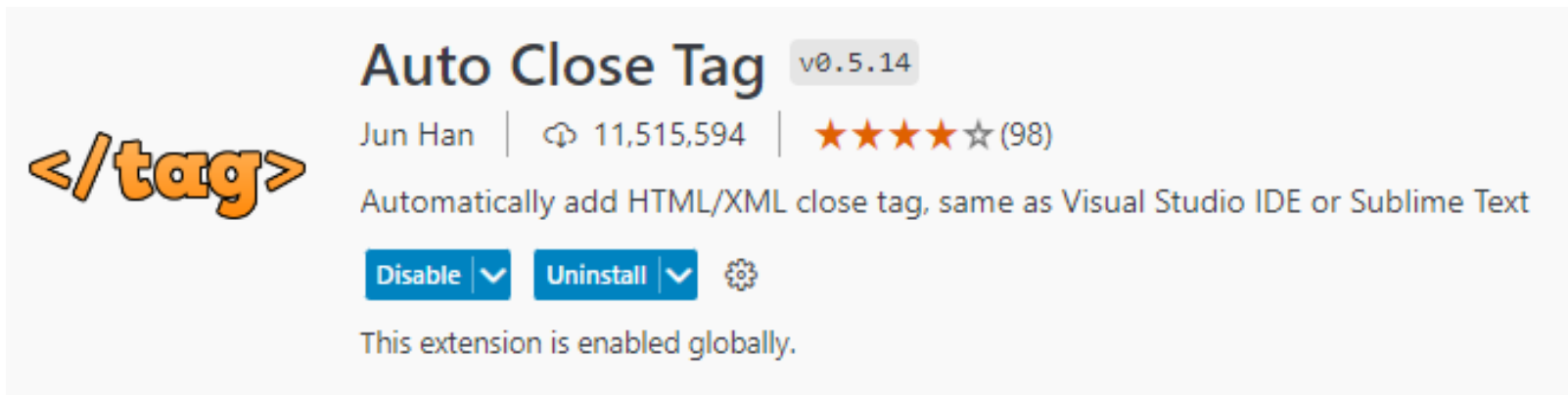
Hay varias extensiones específicas para Odoo. Por ahora instalaremos solo **Odoo Snippets**, que permite generar bloques de código a partir de una palabra clave.



Cierre de etiquetas XML

Parte de la programación en Odoo se hace en ficheros XML en los que puede ser complicado llevar un seguimiento de las etiquetas que hemos abierto y no están cerradas.

Para ayudarnos en esta tarea es muy útil la extensión **Auto Close Tag**, que cierra automáticamente todas las etiquetas que tenemos abiertas

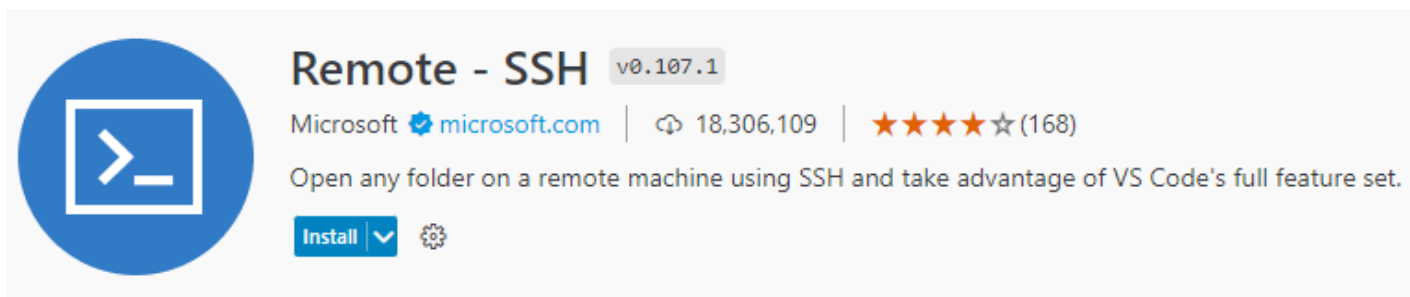


Conexión remota

Si tienes preparado tu entorno con contenedores en Docker sobre Windows no será necesario porque los ficheros de los módulos están en local en tu máquina (recuerda que mapeábamos el directorio `/mnt/extra-addons` a un directorio en la máquina física)

Sin embargo, si tienes tu entorno de desarrollo en una máquina virtual necesitarás conectar tu VS Code (que se ejecuta en Windows) con el directorio de la máquina virtual mediante SSH.

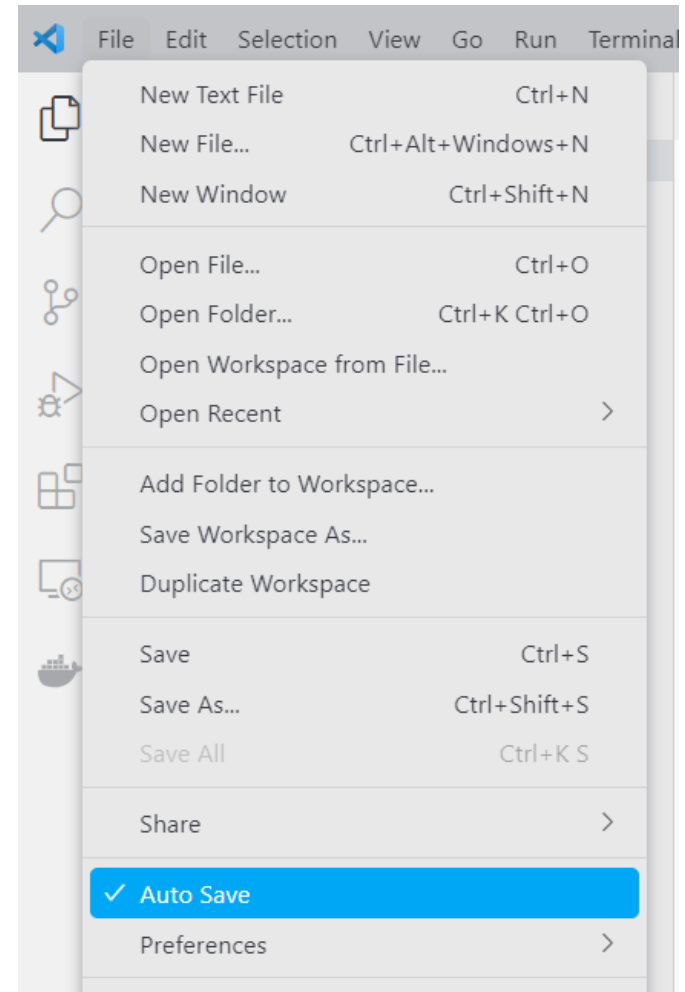
Para ello, necesitarás instalar la extensión **Remote - SSH**, que permite abrir fácilmente carpetas remotas que funcionarán como proyectos en local.



Autoguardado

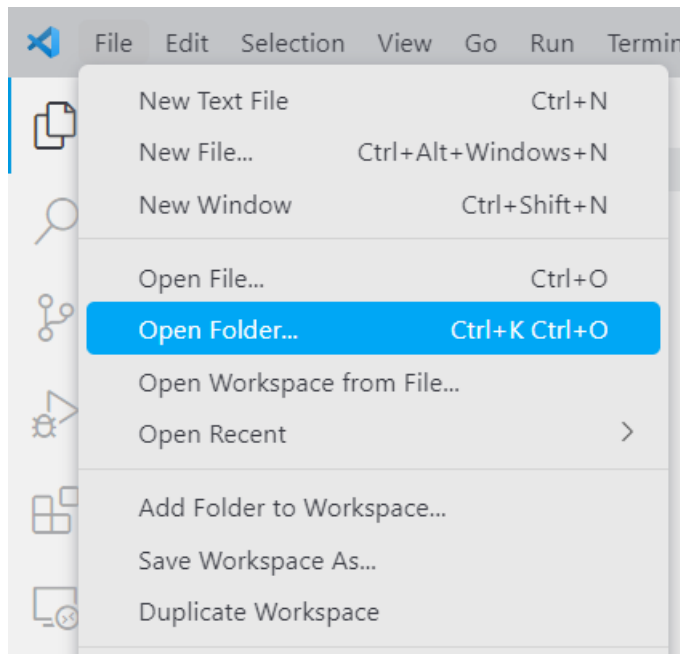
Esta no es una extensión, pero es una funcionalidad muy útil que nos permitirá olvidarnos de tener que guardar los cambios continuamente.

Para activarlos simplemente vamos a *File* -> *Auto Save*, asegurándonos de que quede marcado.



Directorio de trabajo

Ya solo resta abrir el **directorio de trabajo** en *File -> Open Folder*, ya sea sobre el directorio local si tenemos una configuración en contenedores con unidades mapeadas o usando Remote SSH si nuestros datos están en una máquina virtual.



Directorio de trabajo

Solo hemos visto unas pinceladas de Visual Studio Code, pero es una herramienta muy versátil que tiene un gran número de opciones muy útiles cuando estamos programando.

Si quieres profundizar en tu conocimiento de VS Code te aconsejo el curso que tienes en el link de abajo (*requiere registro, pero el curso es gratuito*)

<https://www.udemy.com/course/vscode-mejora-tu-velocidad-para-codificar/>

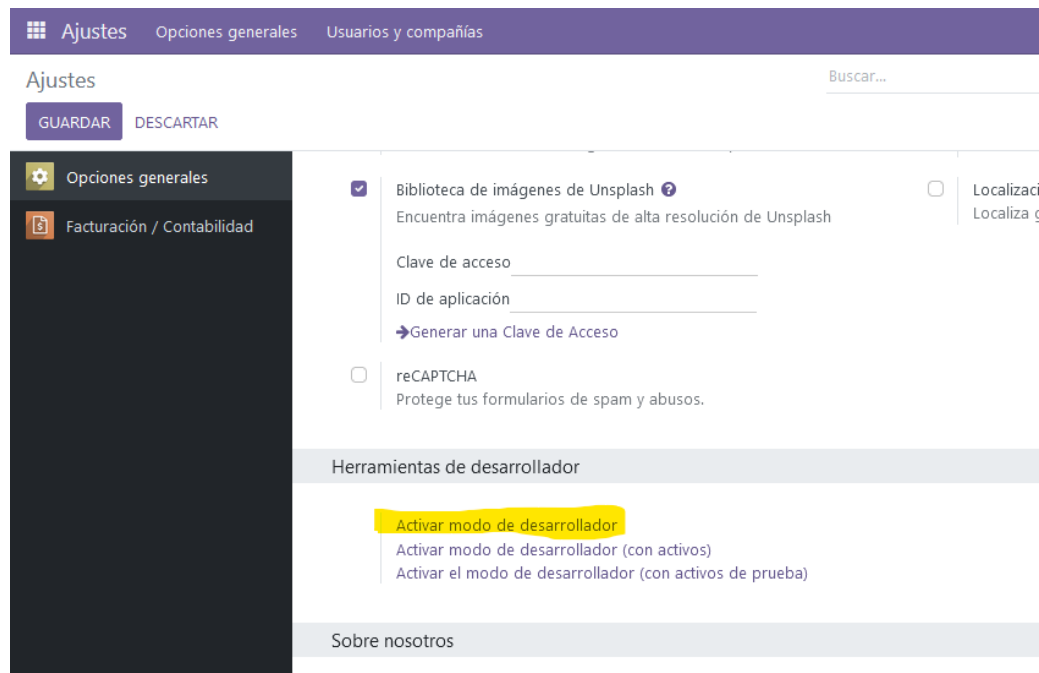


2

PRIMER MÓDULO CON ODOO



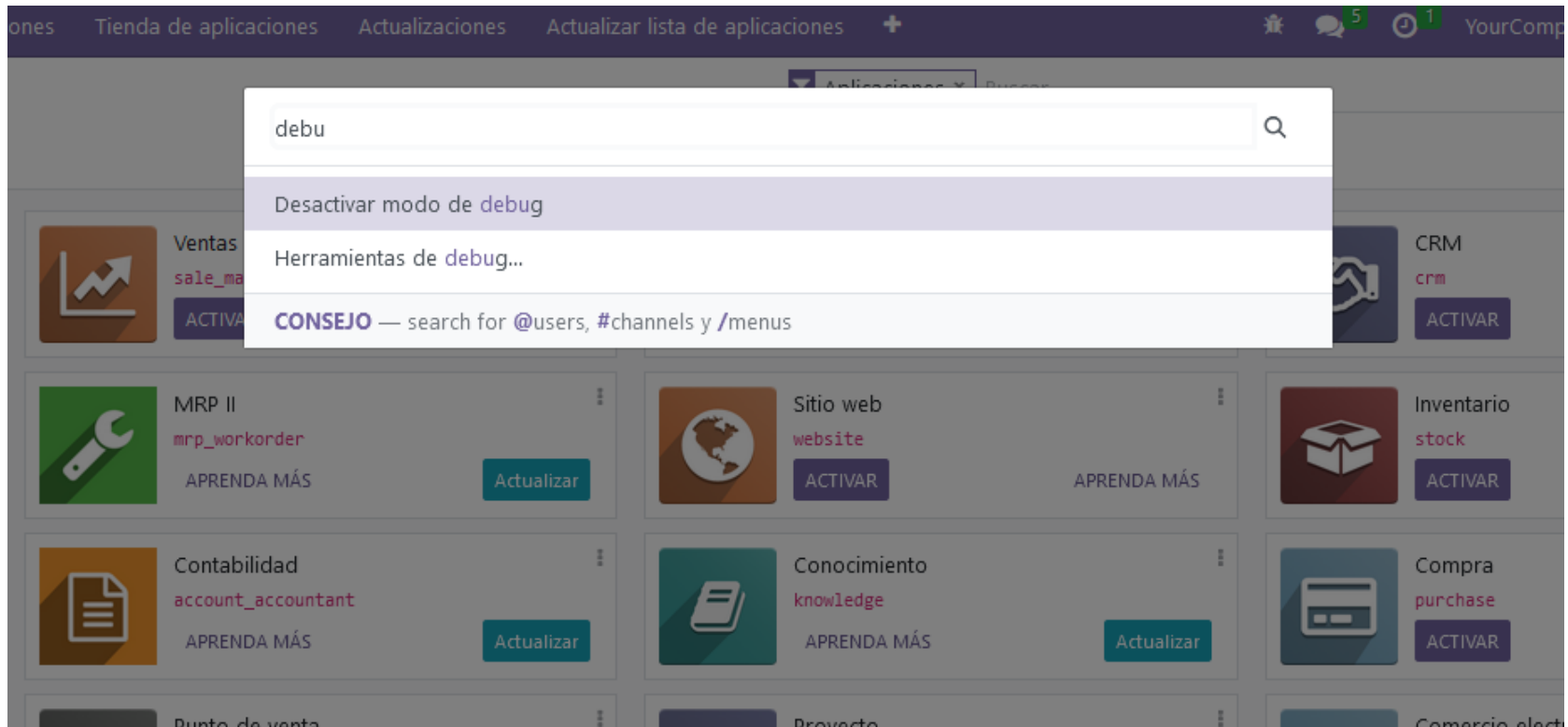
El primer paso será habilitar el **modo desarrollador** en Odoo ([Modo de desarrollador \(modo depuración\) – documentación de Odoo - 16.0](#)) Hay varias formas para hacerlo:



La primera es yendo a *Ajustes* -> *Opciones generales* -> *Herramientas de desarrollador* -> *Activar modo de desarrollador*.


Ten en cuenta que hay que tener instalado por lo menos un módulo para acceder a *Ajustes*.

Otra opción es pulsando Ctrl-K para acceder a la paleta de comandos y buscar **Activar modo de debug**



Hay más opciones que puedes consultar en el enlace anterior.

Otra posibilidad es habilitarlo la extensión disponible tanto para Firefox como para Chrome.

 Firefox Browser
ADD-ONS

[Extensiones](#) [Temas](#) [Más...](#)

[Firefox Add-ons Blog](#) [Taller de extensiones](#) [Central del desarrollador](#) [Log in](#)



Odoo Debug

por [Martin T.](#)

⚠ This add-on is not actively monitored for security by Mozilla. Make sure you trust it before installing.

[Saber más](#)

Toggle debug mode of an Odoo webpage.

- Click for debug.
- Double click for debug with Assets.
- Use ctrl + . for keyboard shortcut

[Agregar a Firefox](#)

6549

Usuarios

6

[Comentarios](#)

★★★★★

4,8 estrellas

5 ★

5

4 ★

1

3 ★

0

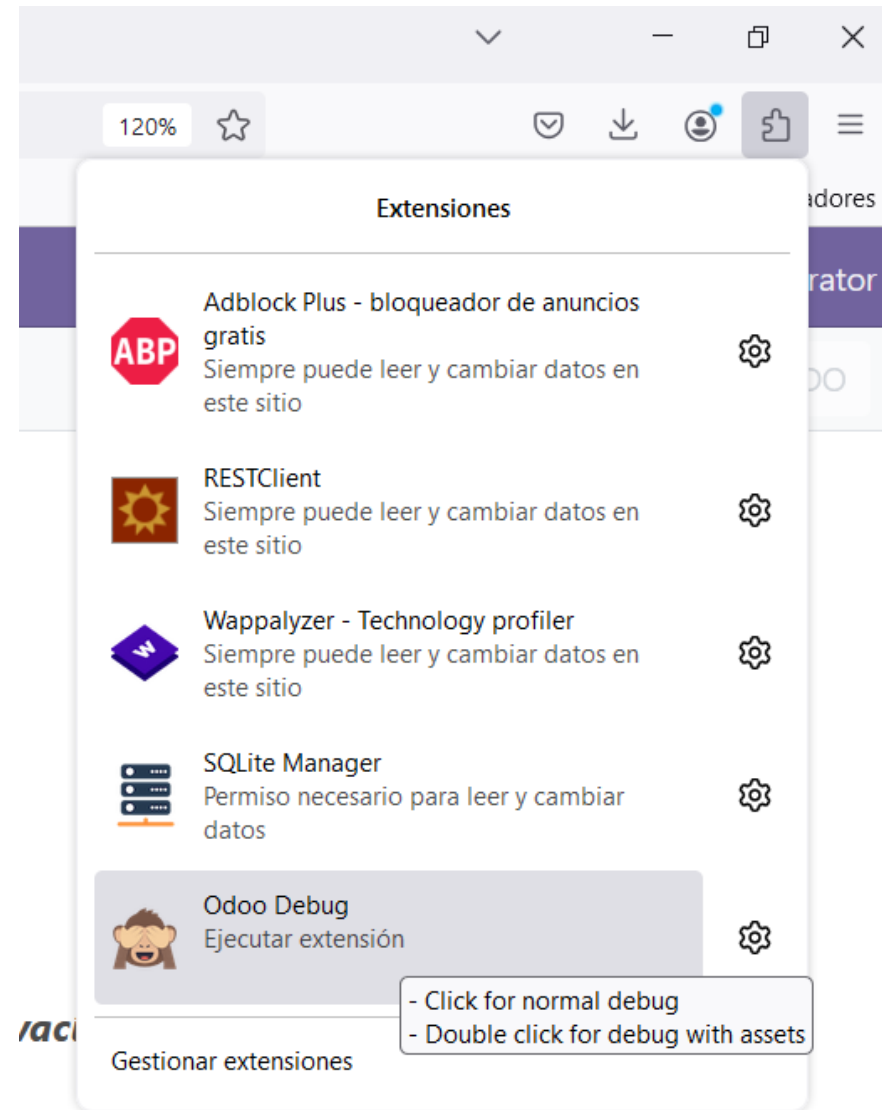
2 ★

0

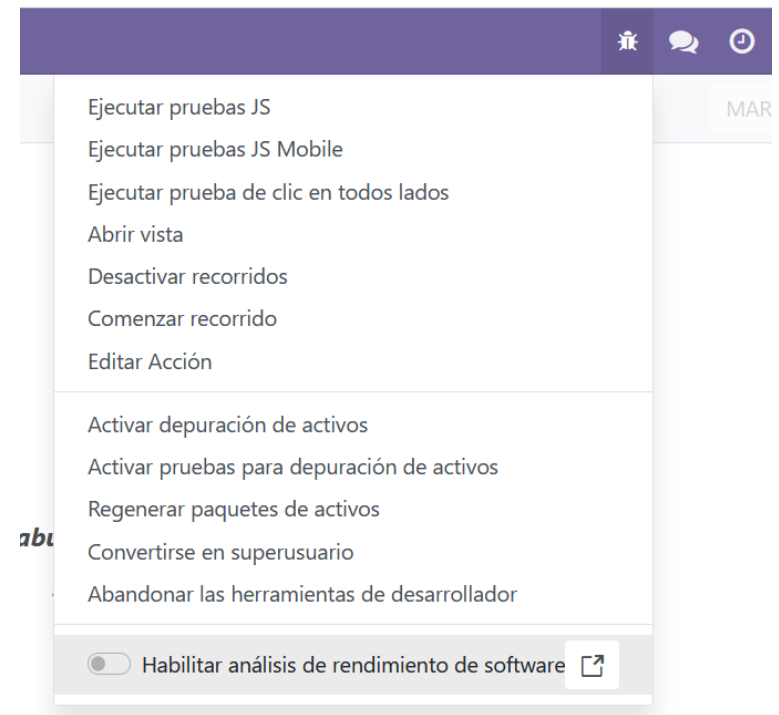
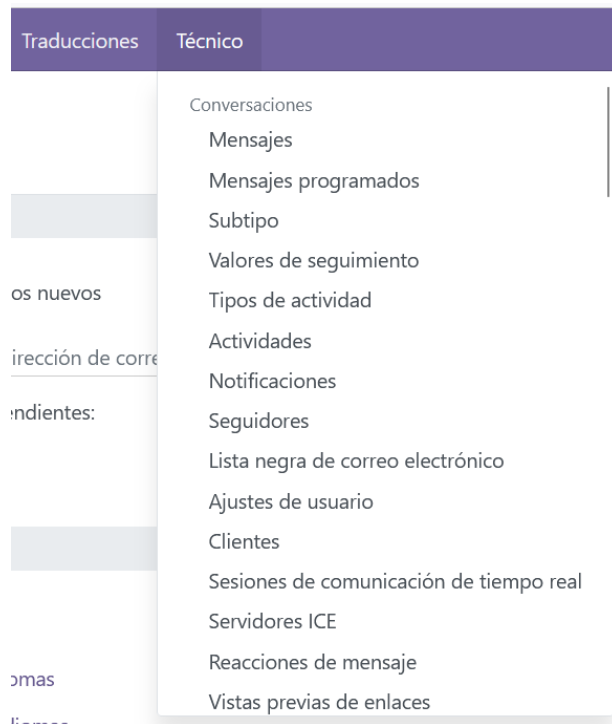
1 ★

0

Una vez cargada la extensión podremos acceder al modo desarrollador simplemente haciendo click en ella.

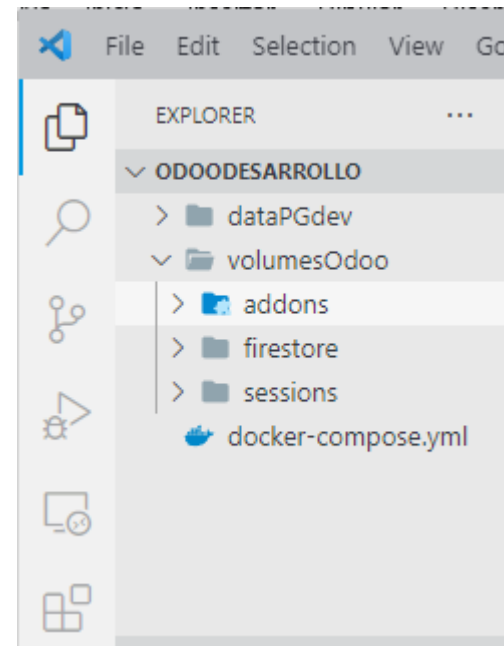
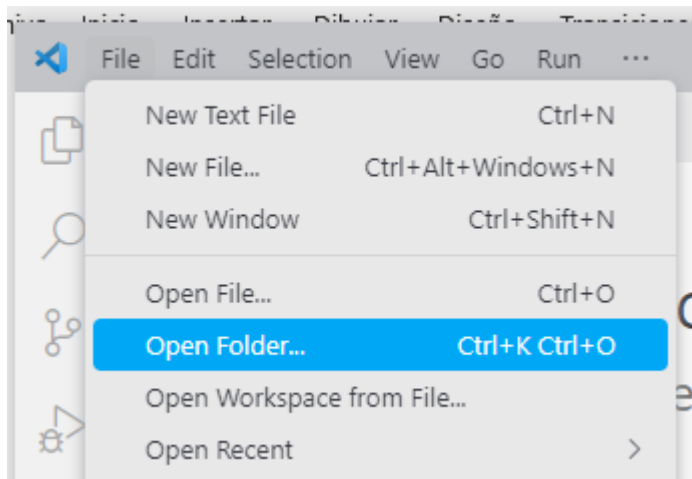


El modo desarrollador habilita varios cambios en nuestra interfaz, como puede ser el **menú depuración** en la parte superior derecha de la pantalla o el **menú técnico** que aparecerá dentro de *Ajustes* y que contiene ajustes avanzados sobre la base de datos

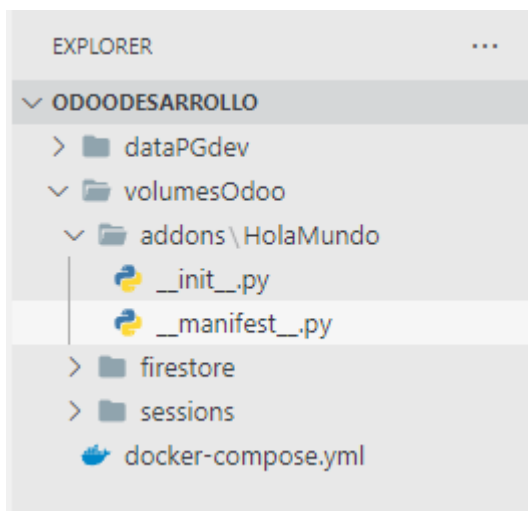
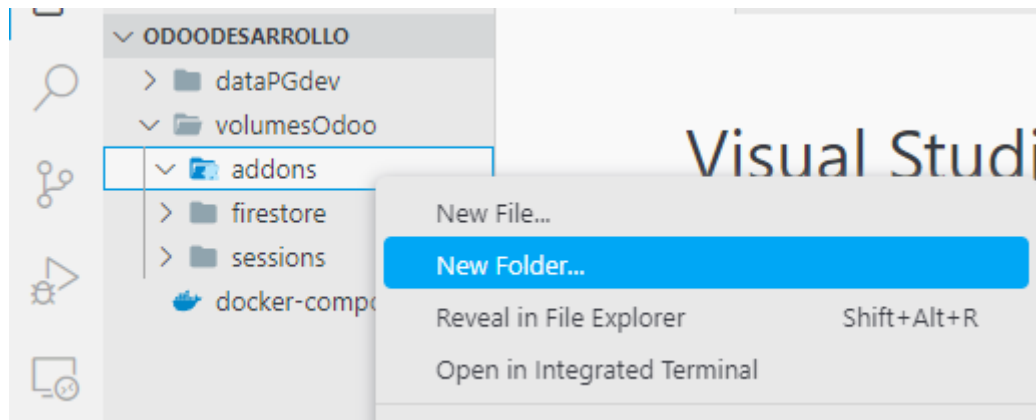


Vamos a crear primero un módulo de prueba, por lo que ya vamos a Visual Studio Code.

Por comodidad, abrimos una carpeta de proyecto. En mi caso he abierto el directorio Odoodesarrollo, desde donde accedo a todas las unidades que hemos mapeado desde el contenedor, incluida **addons**, que es la que nos interesa.



Creamos el directorio que contendrá el *addon*, por ejemplo, la llamamos **HolaMundo**

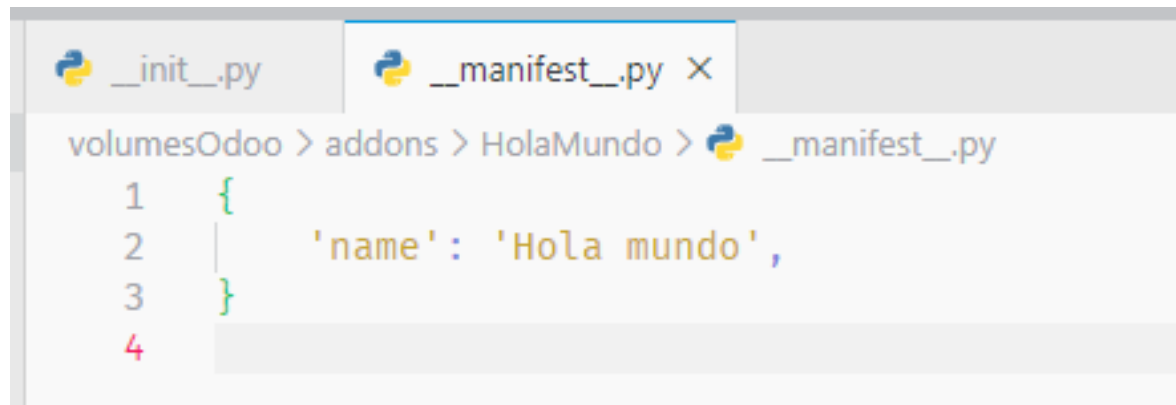


Y dentro del directorio los ficheros `__init__.py` y `__manifest__.py`.

Observa que los guiones bajos son dobles.

Por ahora podemos dejar en blanco el fichero `__init__.py`.

En el otro fichero copiamos el siguiente código:



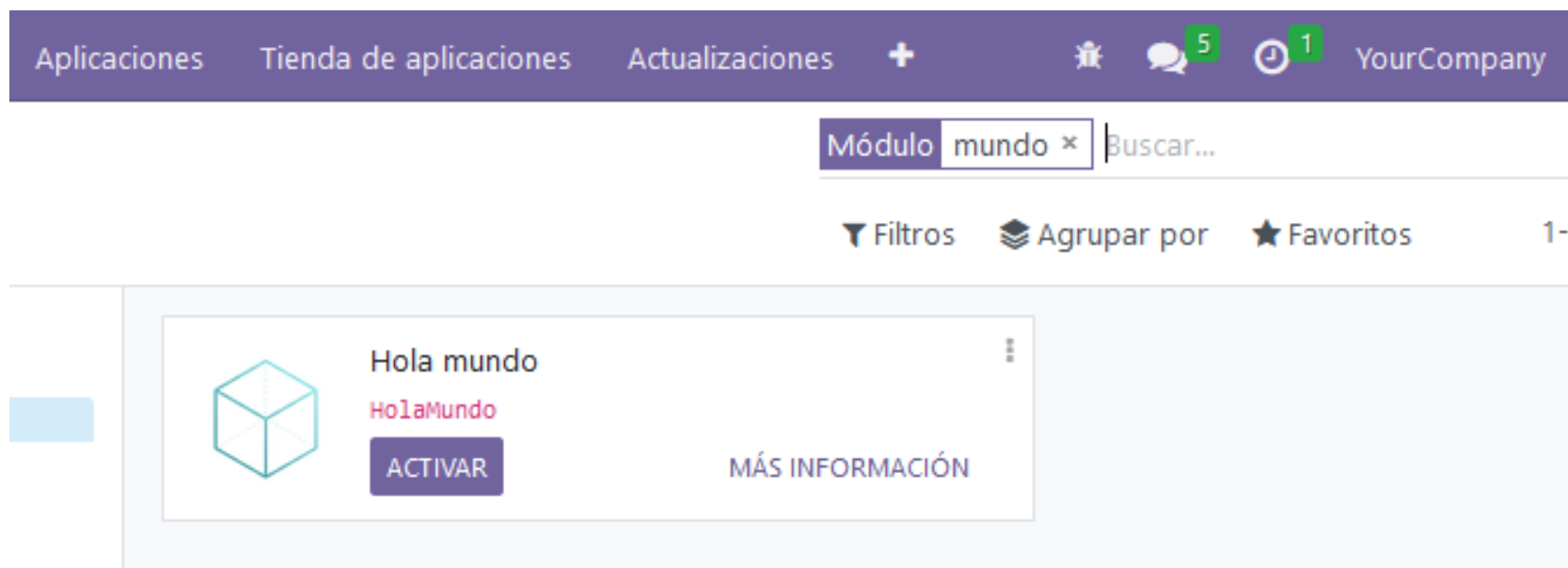
```
__init__.py  __manifest__.py X
volumesOdoo > addons > HolaMundo > __manifest__.py
1  {
2      'name': 'Hola mundo',
3  }
4
```

Ahora ya podemos ir al navegador a localhost:8069 y hacer click en **Actualizar la lista de aplicaciones** para que lea el directorio de addons.

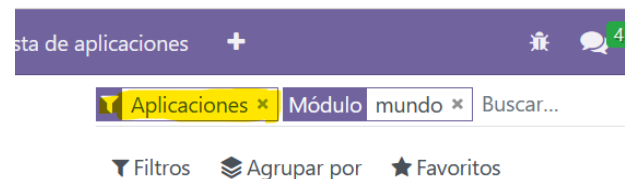
En el cuadro de búsqueda **quitamos** el filtro *Aplicaciones* y ponemos, por ejemplo, *mundo* (o cualquier palabra que tenga el nombre de nuestro módulo)



Si todo ha ido bien, ya tendríamos disponible el módulo que hemos creado.

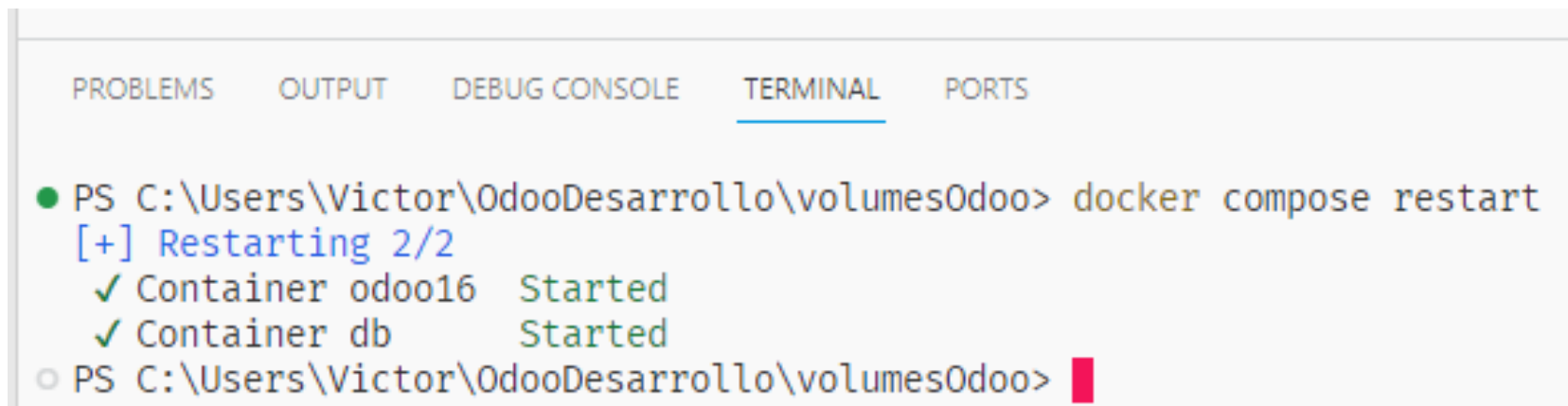


Si no apareciera el módulo y estuviera todo bien podríamos probar a reiniciar el servidor. También asegúrate de quitar el filtro de aplicaciones que sale por defecto.



Si no apareciera el módulo y estuviera todo bien podríamos probar a reiniciar el servidor. Esto lo podemos hacer desde el propio VSCode.

Abrimos una terminal integrada (Ctrl+ñ) y ejecutamos la orden **docker compose restart**, lo que hará que se reinicien todos los contenedores de nuestro proyecto.



```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

● PS C:\Users\Victor\OdooDesarrollo\volumesOdoo> docker compose restart
[+] Restarting 2/2
  ✓ Container odoo16  Started
  ✓ Container db      Started
○ PS C:\Users\Victor\OdooDesarrollo\volumesOdoo>
```

3

ESTRUCTURA DE UN MÓDULO EN ODOO



En el apartado anterior vimos un módulo muy básico, pero implementar un módulo en Odoo requiere de un conjunto de ficheros en **Python** y **XML** organizados en una estructura de directorios.

Aunque se pueden crear manualmente, Odoo proporciona una herramienta que construye directamente todos los ficheros necesarios: **odoo scaffold**

Esta herramienta hay que **ejecutarla en local**. Si tu entorno de desarrollo está con máquinas virtuales no hay ningún problema: abres una sesión SSH y la ejecutas.

Si tienes un entorno con contenedores tienes que acceder mediante la línea de comandos al contenedor Odoo usando el comando:

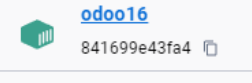
```
docker exec -t -i id_contenedor bash
```

Quiero ejecutar
un comando

Asignar
pseudoterminal
TTY

Mantiene
sesión
interactiva

Puede ser el nombre
o el identificador



Comando
que quiero
ejecutar

Observa que, cuando ejecuto el comando anterior, se me muestra el prompt del Bash. Cualquier comando que introduzca aquí se ejecutará en el contenedor.

Si usamos Docker Compose la orden es similar

```
PS C:\Users\Victor\OdooDesarrollo\volumesOdoo> docker exec -ti odoo16 bash
odoo@841699e43fa4:/$
```

```
PS C:\Users\Victor\Documents\OdooDesarrollo> docker compose exec odoo bash
odoo@e1cfca20fda8:/$
```

Creamos la estructura del módulo con **odoo scaffold**

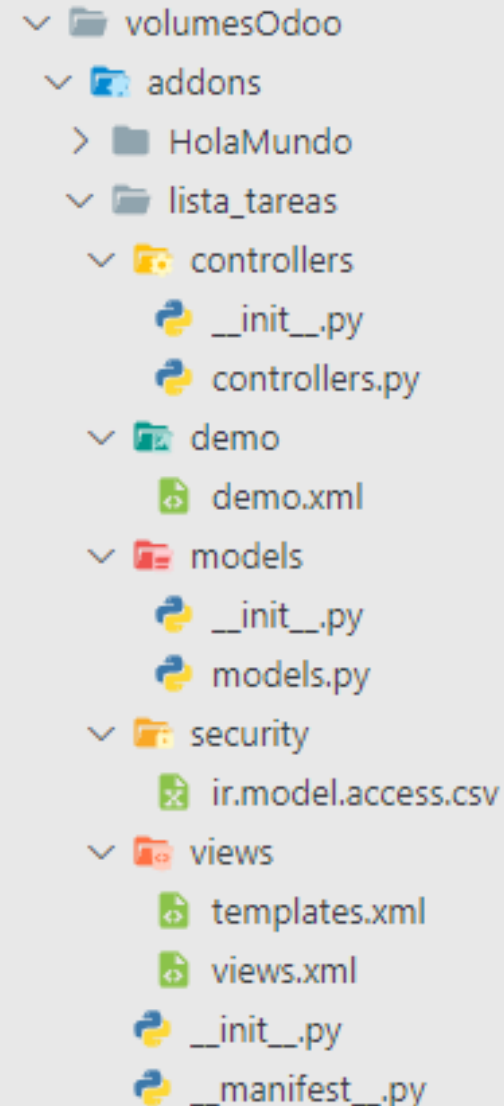
```
odoo@841699e43fa4:~$ odoo scaffold lista_tareas /mnt/extra-addons/
odoo@841699e43fa4:~$
```

Quiero crear la
estructura de un
módulo

El módulo se
llamará
lista_tareas

Se creará en el
directorio de
módulos de Odoo

Recuerda que el directorio `/mnt/extra-addons` lo tenemos mapeado al directorio `~/odooDesarrollo/volumesOdoo/addons`, por lo que desde VSCode ya deberíamos de poder ver lo que ha creado.



```

volumesOdoo
├── addons
│   ├── HolaMundo
│   └── lista_tareas
│       ├── controllers
│       │   ├── __init__.py
│       │   └── controllers.py
│       ├── demo
│       │   └── demo.xml
│       ├── models
│       │   ├── __init__.py
│       │   └── models.py
│       ├── security
│       │   └── ir.model.access.csv
│       └── views
│           ├── templates.xml
│           ├── views.xml
│           ├── __init__.py
│           └── __manifest__.py

```

Los ficheros generados son:

- **models/models.py**: define un ejemplo del modelo de datos y sus campos
- **views/views.xml**: describe las vistas de nuestro módulo
- **demo/demo.xml**: incluye datos "demo" para el ejemplo propuesto de modelo
- **controllers/controllers.py**: contiene un ejemplo de controlador de rutas
- **views/templates.xml**: contiene dos ejemplos de vistas "qweb" usado por el controlador de rutas.
- **__manifest__.py**: incluye información como el título, descripción, ficheros a cargar, ...

Como estamos trabajando en **modo desarrollo** (opción `--dev=all`), el servidor lee cada vez las vistas, datos y código Python directamente de los ficheros, por lo que no es necesario reiniciar el servicio ni actualizar el módulo para ver los cambios.

En un **entorno de producción** la cosa es diferente:

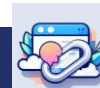
- Las vistas, datos,... (todo lo que esté en XML) se guardan en la base de datos al instalar el módulo, así que solo se actualiza cuando se instala o actualiza el módulo.
- Los ficheros Python son cargados cada vez que se inicia el servicio Odoo, por lo que, para que se apliquen los cambios hay dos opciones: recargar el módulo o reiniciar el servicio Odoo.

Vamos a ver cómo haríamos un módulo que gestione una lista de tareas.

Comenzamos introduciendo los datos en el archivo `__manifest__.py`. Los campos más relevantes son:

- **name:** nombre que tendrá el módulo
- **summary** y **description:** resumen y descripción del módulo
- **author** y **website:** autor y página web del módulo
- **application:** si se considera una aplicación completa o es un módulo técnico que proporciona funcionalidad extra a otra aplicación.
- **category:** categoría a la que pertenece el módulo
- **version:** debe seguir las normas de versionado semántico (<https://semver.org/>)
- **depends:** listado de módulos de los que depende y que, por tanto, se tienen que cargar antes.
- **data:** lista de ficheros de datos que se tienen que instalar o actualizar con el módulo

<https://www.odoo.com/documentation/16.0/es/developer/reference/backend/module.html>



```
volumesOdoo > addons > lista_tareas >  __manifest__.py
```

```
1  # -*- coding: utf-8 -*-
2  {
3      'name': "lista_tareas",
4      'summary': ""
5      |     Módulo para la gestión de una lista de tareas"",
6      'description': ""
7      |     Módulo que permite crear tareas con una prioridad asociada. Permite
8      |     marcar tareas como realizadas o no realizadas y tiene un campo
9      |     calculado llamado 'Urgente' que se marca si la prioridad es superior a
10     |     10
11     |     "",
12     'author': "Sergi García",
13     'website': "https://github.com/sergarb1/ApuntesSistemasGestionEmpresarial",
14     'application': True,
15     'category': 'Productivity',
16     'version': '0.1',
17     'depends': ['base'],
18     'data': [
19         |     'security/ir.model.access.csv',
20         |     'views/views.xml'
21     ]
22 }
```

Ahora vamos a por el **modelo**, que está en el fichero **models.py**

```
__manifest__.py  models.py 1 X
1  # -*- coding: utf-8 -*-
2
3  from odoo import models, fields, api
4
5  class lista_tareas(models.Model):
6      _name_ = 'lista_tareas.lista_tareas'
7      _description = "lista_tareas.lista_tareas"
8
9      tarea = fields.Char()
10     prioridad = fields.Integer()
11     urgente = fields.Boolean( compute='_value_urgente', store=True )
12     realizada = fields.Boolean()
13
14     @api.depends('prioridad')
15     def _value_urgente(self):
16         for record in self:
17             record.urgente = record.prioridad>10
18
```

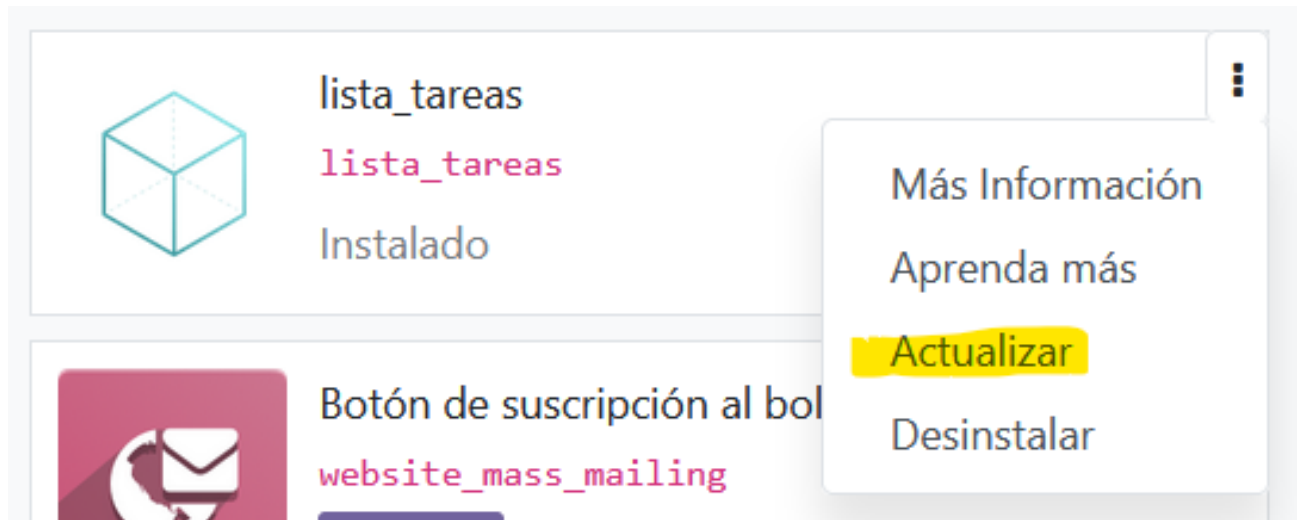
Y la **vista** en el fichero **views.xml**

```
volumesOdoo > addons > lista_tareas > views >  views.xml
1  <?xml version='1.0' encoding='utf-8'?>
2  <odoo>
3      <data>
4          <!-- explicit list view definition -->
5          <record model="ir.ui.view" id="lista_tareas.list">
6              <field name="name">lista_tareas list</field>
7              <field name="model">lista_tareas.lista_tareas</field>
8              <field name="arch" type="xml">
9                  <tree>
10                     <field name="tarea" />
11                     <field name="prioridad" />
12                     <field name="urgente" />
13                     <field name="realizada" />
14                 </tree>
15             </field>
16         </record>
17
```

Continúa en la siguiente página....

```
17
18  <!-- actions opening views on models -->
19  <record model="ir.actions.act_window" id="lista_tareas.action_window">
20    <field name="name">Listado de tareas pendientes</field>
21    <field name="res_model">lista_tareas.lista_tareas</field>
22    <field name="view_mode">tree,form</field>
23  </record>
24
25  <!-- top menu item -->
26  <menuitem name="Listado de tareas" id="lista_tareas.menu_root" />
27
28  <!-- menu categories -->
29  <menuitem name="Opciones Lista Tareas"
30    | | | | id="lista_tareas.menu_1"
31    | | | | parent="lista_tareas.menu_root" />
32
33  <!-- actions -->
34  <menuitem name="Mostrar lista"
35    | | | | id="lista_tareas.menu_1_list"
36    | | | | parent="lista_tareas.menu_1"
37    | | | | action="lista_tareas.action_window" />
38  </data>
39  </odoo>
```

El módulo se debería actualizar automáticamente (si estás en modo desarrollador), si no lo hiciera puedes actualizarlo manualmente




Y ya tendríamos el módulo, si vamos a *Más información* podremos ver toda la información que pusimos en el fichero `__manifest__.py`

lista_tareas

Por Sergi García


ACTIVAR



lista_tareas

lista_tareas

Instalado



Botón de suscripción al bol

website_mass_mailing

ACTIVAR

Más Información

Aprenda más

Actualizar

Desinstalar

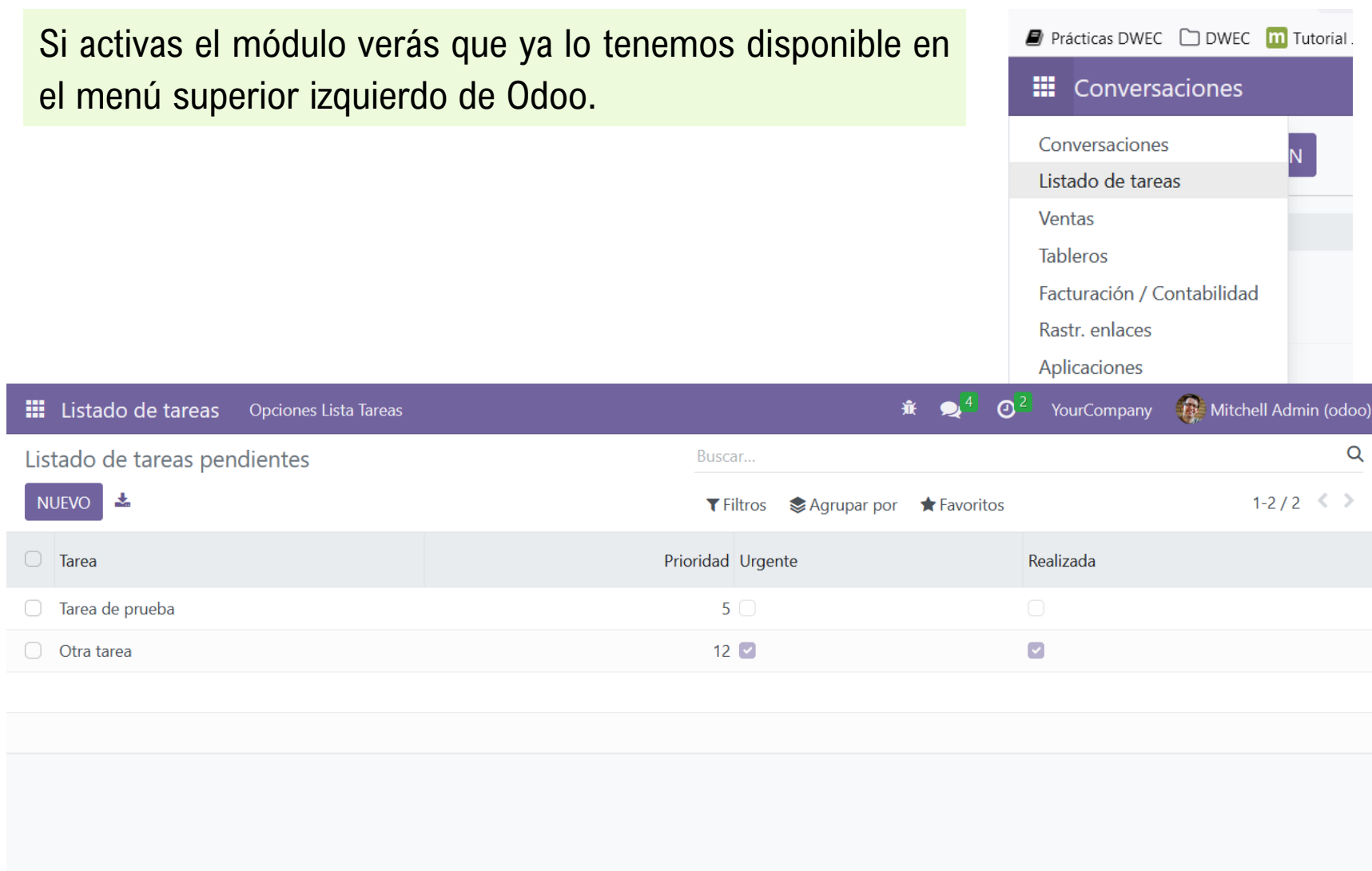
MÁS INFORMACIÓN

Información

Datos técnicos

Sitio web ?	https://github.com/sergarb1/ApuntesSistemasGestionEmpresarial	Nombre técnico ?	lista_tareas
Categoría ?	Productividad	Licencia ?	LGPL versión 3
Resumen ?	Short (1 phrase/line) summary of the module's purpose, used as subtitle on modules listing or apps.openerp.com		
		Última versión ?	16.0.0.1

Si activas el módulo verás que ya lo tenemos disponible en el menú superior izquierdo de Odoo.



The screenshot displays the Odoo web interface. At the top, a navigation bar includes links for 'Prácticas DWEC', 'DWEC', and 'Tutorial'. A dropdown menu is open, showing 'Conversaciones' (highlighted), 'Listado de tareas', 'Ventas', 'Tableros', 'Facturación / Contabilidad', 'Rastr. enlaces', and 'Aplicaciones'. Below this, a purple header bar contains 'Listado de tareas' and 'Opciones Lista Tareas'. The main content area is titled 'Listado de tareas pendientes' and features a search bar, a 'NUEVO' button, and filters. A table lists tasks with columns for 'Tarea', 'Prioridad', 'Urgente', and 'Realizada'.

Tarea	Prioridad	Urgente	Realizada
<input type="checkbox"/> Tarea de prueba	5	<input type="checkbox"/>	<input type="checkbox"/>
<input type="checkbox"/> Otra tarea	12	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>

Ahora que ya has conseguido hacer funcionar tu primer módulo en Odoo, vamos a ver exactamente qué hace el código que hemos escrito.

Fichero `models.py`

Odoo se basa en el sistema **ORM (Object Relational Mapping)**, una técnica que consiste en mapear directamente las clases creadas en un lenguaje de programación sobre una base de datos relacional.

De esta forma, no necesitamos crear la base de datos como tal, sino que simplemente creamos una clase con las propiedades que queramos y, automáticamente, se crearán los campos correspondientes en la base de datos.

Para que esto sea posible, la clase que creamos deberá heredar de la clase **`models.Model`**

```
class lista_tareas(models.Model):  
    _name = 'lista_tareas.lista_tareas'  
    _description = 'lista_tareas.lista_tareas'
```

En Python la herencia se indica así. Esto quiere decir que la clase **lista_tareas** hereda de la clase **models.Model**

_description es el nombre general para el modelo que se usa, por ejemplo, para búsquedas.

```
class lista_tareas(models.Model):  
    _name = 'lista_tareas.lista_tareas'  
    _description = 'lista_tareas.lista_tareas'
```

La propiedad **_name** determina el nombre de la tabla que se va a crear. Aquí vemos desde **psql** la tabla que ha creado automáticamente

```
odoo=# \dt lista_tareas_lista_tareas  
List of relations  
Schema | Name | Type | Owner  
-----+-----+-----+-----  
public | lista_tareas_lista_tareas | table | odoo  
(1 row)
```

<https://www.odoo.com/documentation/17.0/developer/reference/backend/orm.html#fields>



Creamos una variable por cada registro que va a tener la tabla en la base de datos. El objeto **fields** tiene diversas funciones según el tipo de datos que queramos. Más información en el enlace

```
odoo=# \d lista_tareas_lista_tareas
```

Column	Type	Collation	Nullable	
id	integer		not null	nextval('lista_
prioridad	integer			
create_uid	integer			
write_uid	integer			
tarea	character varying			
urgente	boolean			
realizada	boolean			
create_date	timestamp without time zone			
write_date	timestamp without time zone			

```
tarea = fields.Char()
prioridad = fields.Integer()
urgente = fields.Boolean( compute='_value_urgente', store=True )
realizada = fields.Boolean()
```

Podemos agregar información extra para cada registro. Por ejemplo, aquí estamos indicando que el campo *urgente* es un **campo calculado** mediante la función *_value_urgente*

Con *store* indicamos que se almacenará en la base de datos (sino se recalculará cada vez que sea requerido)

Este **decorador** indica las dependencias del campo calculado para saber cuándo hay que recalcularlo.

La función para el campo calculado

```
@api.depends('prioridad')
def _value_urgente(self):
    for record in self:
        if record.prioridad > 10:
            record.urgente = True
        else:
            record.urgente = False
```

Ya veremos la sintaxis de Python, pero algo importante es que el **indentado** sirve para establecer los bloques de código

Este campo calculado (urgente) se calcula como verdadero si la prioridad es superior a 10 y falso en caso contrario.

Fichero `views.xml`

Este fichero en formato XML determina qué es lo que va a ver el usuario.

```
<?xml version='1.0' encoding='utf-8'?>
<odoo>
  <data>
    <!-- explicit list view definition -->
    <record model="ir.ui.view" id="lista_tareas.list">
      <field name="name">lista_tareas list</field>
      <field name="model">lista_tareas.lista_tareas</field>
      <field name="arch" type="xml">
```

Todo está dentro de la etiqueta `<odoo>`

El modelo `ir.ui.view` es un modelo interno de Odoo que se utiliza en las vistas donde se muestra un campo o una lista de árbol (tree list)

Indicamos el modelo que corresponde a esta vista (recuerda que es el nombre que le dimos en `models.py`)

```
class lista_tareas(models.Model):
    _name = 'lista_tareas.lista_tareas'
    _description = 'lista_tareas.lista_tareas'
```

Si listamos las tablas (modelos) de Odoo veremos que destacan dos grupos de tablas: las que comienzan por **ir** (Information Repository) y las que comienzan por **res** (Resource).

- Los **recursos** corresponden a algo del "mundo real" que almacenamos en Odoo, por ejemplo, países (`res_country`) o empresas (`res_company`)
- Los **Information Repository** son usados para almacenar datos necesarios para Odoo para saber cómo trabajar como aplicación. Por ejemplo, para definir menús (`ir_ui_menu`) o vistas (`ir_ui_view`)

```
public | ir_act_client
public | ir_act_report_xml
public | ir_act_server
public | ir_act_server_group_rel
public | ir_act_server_res_partner_rel
public | ir_act_url
public | ir_act_window
public | ir_act_window_group_rel
public | ir_act_window_view
public | ir_actions
public | ir_actions_todo
public | ir_asset
public | ir_attachment
public | res_bank
public | res_company
public | res_company_users_rel
public | res_config
public | res_config_installer
public | res_config_settings
public | res_country
public | res_country_group
public | res_country_group_pricelist_rel
public | res_country_res_country_group_rel
public | res_country_state
public | res_currency
public | res_currency_rate
```

```
odoo=# select id,code,name from res_country;
 id | code | name
-----+-----+-----
212 | SZ    | {"en_US": "Eswatini"}
 3 | AF    | {"en_US": "Afghanistan", "es_ES": "Afganistán"}
 6 | AL    | {"en_US": "Albania", "es_ES": "Albania"}
62 | DZ    | {"en_US": "Algeria", "es_ES": "Argelia"}
11 | AS    | {"en_US": "American Samoa", "es_ES": "Samoa Americana"}
 1 | AD    | {"en_US": "Andorra", "es_ES": "Andorra"}
 8 | AO    | {"en_US": "Angola", "es_ES": "Angola"}
 5 | AI    | {"en_US": "Anguilla", "es_ES": "Anguilla"}
 9 | AQ    | {"en_US": "Antarctica", "es_ES": "Antártida"}
40 | CF    | {"en_US": "Central African Republic", "es_ES": "República Centro Africana"}
214 | TD    | {"en_US": "Chad", "es_ES": "Chad"}
```

Ejemplo del
contenido de
`res_country`


```
<field name="arch" type="xml">
  <tree>
    <field name="tarea" />
    <field name="prioridad" />
    <field name="urgente" />
    <field name="realizada" />
  </tree>
</field>
</record>
```

La etiqueta **<tree>** define cómo va a ser la vista: el **tree view** o **list view** muestra múltiples registros en forma de lista donde cada fila representa un registro de la base de datos

Indico qué campos quiero que se muestren

```
odoo=# select * from lista_tareas_lista_tareas;
 id | prioridad | create_uid | write_uid | tarea          | urgente | realizada |
-----+-----+-----+-----+-----+-----+-----+
  1 |         0 |          2 |          2 | Primera tarea | f       | t         |
  2 |         6 |          2 |          2 | Segunda tarea | f       | f         |
(2 rows)
```

Lo que hay en la base de datos y cómo lo vemos

Listado de tareas pendientes			
NUEVO 		Buscar...	
		Filtros Agrupar por Favoritos	
<input type="checkbox"/> Tarea	Prioridad	Urgente	Realizada
<input type="checkbox"/> Primera tarea	0	<input type="checkbox"/>	<input checked="" type="checkbox"/>
<input type="checkbox"/> Segunda tarea	6	<input type="checkbox"/>	<input type="checkbox"/>

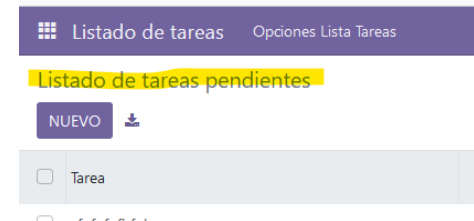
Una **acción** define el comportamiento del sistema como respuesta a las acciones del usuario.

Las *Window Actions* son el tipo más común de acción, se usa para presentar la visualización del modelo a través de las vistas. En este ejemplo indicamos que el modelo se presentará con las vistas **tree** y **form**

```
<!-- actions opening views on models -->
<record model="ir.actions.act_window" id="lista_tareas.action_window">
  <field name="name">Listado de tareas pendientes</field>
  <field name="res_model">lista_tareas.lista_tareas</field>
  <field name="view_mode">tree,form</field>
</record>
```

El nombre que se mostrará

Este es el modelo al que corresponden las vistas




```
←!— actions opening views on models —→  
<record model="ir.actions.act_window" id="lista_tareas.action_window">  
  <field name="name">Listado de tareas pendientes</field>  
  <field name="res_model">lista_tareas.lista_tareas</field>  
  <field name="view_mode">tree,form</field>  
</record>
```

La vista **tree** es la que muestra una serie de filas con los datos del modelo

Listado de tareas pendientes

NUEVO

Buscar...

Filtros Agrupar por Favoritos 1-2 / 2

<input type="checkbox"/> Tarea	Prioridad	Urgente	Realizada
<input type="checkbox"/> sfgfgfghfgh	0	<input type="checkbox"/>	<input type="checkbox"/>
<input type="checkbox"/> cvbcb	0	<input type="checkbox"/>	<input type="checkbox"/>

La vista **form** se usa habitualmente para recoger datos introducidos por el usuario

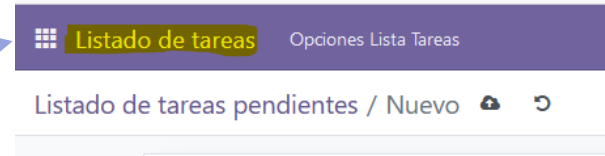
Listado de tareas pendientes / Nuevo

Acción

Tarea ?	Una tarea
Urgente ?	<input type="checkbox"/>
Prioridad ?	7
Realizada ?	<input type="checkbox"/>

Definimos los **menús** que se mostrarán en la vista

Esta es la **raíz** del menú



```

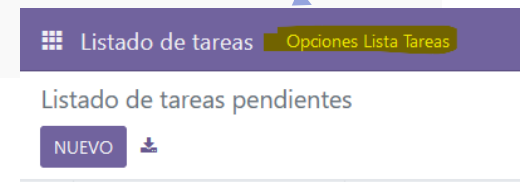
<!-- top menu item -->
<menuitem name="Listado de tareas" id="lista_tareas.menu_root" />

<!-- menu categories -->
<menuitem name="Opciones Lista Tareas"
  id="lista_tareas.menu_1"
  parent="lista_tareas.menu_root" />

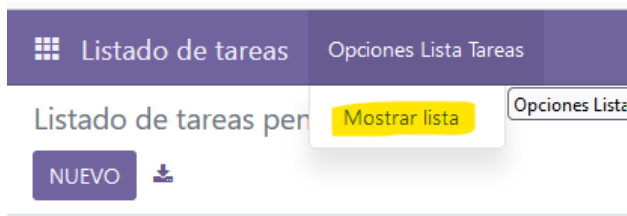
<!-- actions -->
<menuitem name="Mostrar lista"
  id="lista_tareas.menu_1_list"
  parent="lista_tareas.menu_1"
  action="lista_tareas.action_window" />

```

Definimos cada uno de los **submenús**



Por último, los **elementos** de cada menú



4

NORMAS DE NOMENCLATURA



Como ya has visto, en Odoo hay que hacer referencia a múltiples elementos mediante su nombre o identificador, por lo que es sumamente importante seguir unas **normas de nomenclatura** para evitar caer en inconsistencias y, por tanto, en errores al ejecutar nuestro módulo.

Nombre del módulo

Utiliza nombres en **minúsculas** y **separados por guiones bajos**.

El nombre debe ser descriptivo y breve

Ejemplo: `project_management`, `task_list`

Nombre del modelo

Los nombres de los modelos siguen la convención:

`nombre_modulo.nombre_modelo`

Usa minúsculas y guiones bajos para separar las palabras

Ejemplo: `library_management.library_author`

Campos del modelo

También en minúsculas y con guiones bajos

Usa nombres descriptivos y evita abreviaturas a menos que sean estándar

Evita caracteres no anglosajones

Ejemplos:

- `name`: nombre del registro
- `order_date`: fecha de pedido

Identificadores de registros XML

Usa un prefijo relacionado con el módulo seguido de un identificador del propósito

Usa minúsculas, guiones bajos y nombres descriptivos

Ejemplos:

- `view_project_task_form`: vista de formulario para el modelo `project_task`
- `action_project_task_tree`: vista para la vista lista de `project.task`
- `menu_project_taks_root`: menú raíz del módulo anterior

Acciones

Las acciones, tanto de ventana como de servidor, deben usar nombres descriptivos que indiquen el modelo y el propósito.

Ejemplos:

- `action_project_task_tree`: acción que muestra la vista lista del modelo
- `action_invoice_send_mail`: acción de servidor del modelo invoice que envía un correo electrónico.

