

Sistemas de Gestión Empresarial

# **UD4 Entorno de desarrollo y primer módulo Odoo**

---




**Reconocimiento – NoComercial – CompartirIgual (BY-NC-SA):** No se permite un uso comercial de la obra original ni de las posibles obras derivadas, la distribución de las cuales se debe hacer con una licencia igual a la que regula la obra original.

## Nomenclatura

A lo largo de este tema se utilizarán distintos símbolos para distinguir elementos importantes dentro del contenido. Estos símbolos son:

 **Importante**

 **Atención**

 **Interesante**

## ÍNDICE DE CONTENIDO

1.	Introducción	3
1.1	¿Cómo vamos a orientar esta unidad?	3
2.	Creando un directorio para de módulos	6
2.1	Instalación manual	6
2.2	Instalación mediante Docker y Docker Compose	6
3.	Entorno de desarrollo	7
3.1	Visual Studio Code	7
3.2	Control de versiones usando Git + Visual Studio Code	7
3.3	PyCharm	8
3.4	Control de versiones usando Git + PyCharm	8
4.	Activando el modo desarrollador en Odoo 17	8
5.	Nuestro primer módulo: “Hola mundo”	8
6.	Creando módulos en Odoo	9
6.1	Creando módulos con “Odoo Scaffold”	9
6.2	Estructura de un módulo Odoo	11
6.3	Módulos en producción	12
6.4	Módulos en desarrollo	13
7.	Ejemplo de módulo “Lista de tareas”	13
8.	Bibliografía	16

## 1. INTRODUCCIÓN

En la unidad 2 se explicaba cómo instalar Odoo, tanto con una configuración pensada para un entorno de producción como una configuración pensada para un entorno de desarrollo. En esta unidad vamos a insistir un poco más en la configuración del entorno de desarrollo, incluyendo tanto el propio Odoo, como el IDE de desarrollo con sus respectivos plugins. Con todo esto listo, realizaremos nuestros primeros módulos de Odoo.

### 1.1 ¿Cómo vamos a orientar esta unidad?

En primer lugar, preparamos todo el entorno de desarrollo. Con un sistema ERP Odoo instalado, se pueden desarrollar módulos que amplían su funcionalidad sin una gran preparación del entorno. Tan sólo una terminal y un editor de texto son suficientes para poder desarrollar con Odoo.

No obstante, se recomienda realizar una configuración previa tanto del propio sistema ERP Odoo como de las herramientas de desarrollo asociadas para hacer el trabajo más fácil y aumentar la productividad.

Una vez configurado el entorno de desarrollo, comenzaremos a introducir los primeros módulos de Odoo. Comenzaremos con un sencillo módulo “que no hace nada”, pero que ya se puede instalar. Este módulo nos ayudará a validar si tenemos una configuración correcta de nuestro entorno de desarrollo.

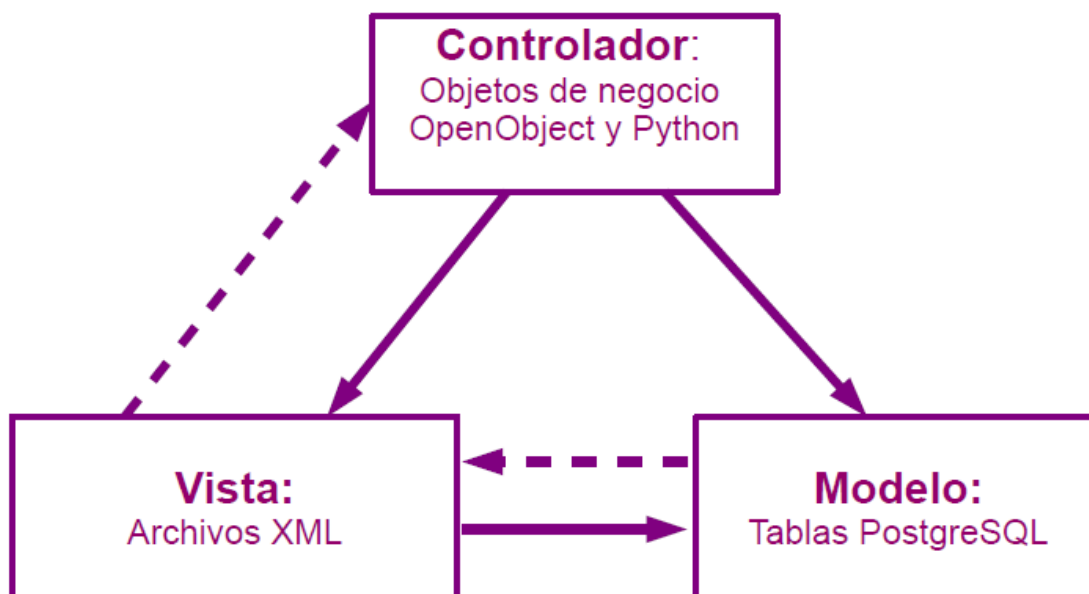
A continuación, crearemos ampliaremos el módulo base anterior, añadiendo cosas para aumentar su funcionalidad. Esa será la metodología que seguiremos en esta unidad de introducción al desarrollo de módulos.

El desarrollo de módulos de Odoo puede llegar a ser muy complejo y sólo los programadores expertos son capaces de profundizar desde el principio sin ver cómo va funcionando. En la siguiente unidad ya profundizaremos en mayor detalle en la programación de módulos para Odoo.

Odoo utiliza un modelo vista-controlador

Las principales características de la programación en Odoo son:

- Openobject como framework.
- Se enmarca dentro de la programación tipo RAD (Rapid application Development)
- Emplea un modelo ORM (Object Relational Mapping). ORM te permite convertir los datos de tus objetos en un formato correcto para poder guardar la información en una base de datos (mapeo) creándose una base de datos virtual donde los datos que se encuentran en nuestra aplicación, quedan vinculados a la base de datos (persistencia).
- Modelo vista controlador.
- Flujos de trabajo o workflows.
- Diseñador de informes.
- Facilita las traducciones.



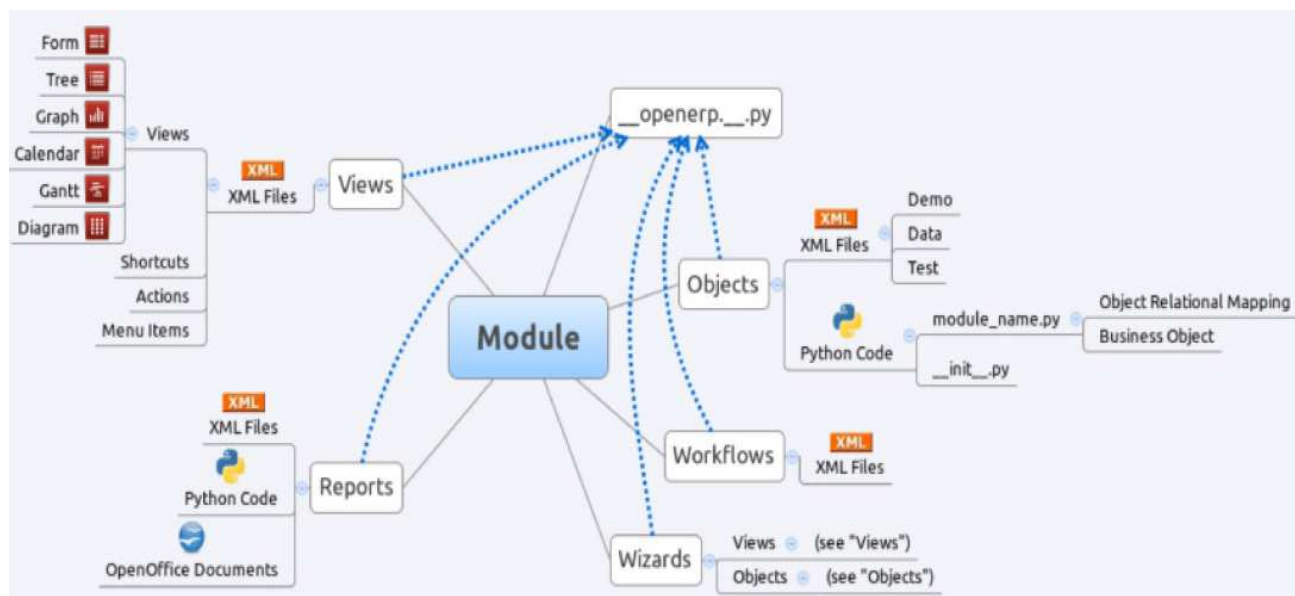
Las principales características de la base de datos son:

- No hay un diseño explícito de la base de datos.
- El ERP mapea el diseño del SGBD PostgreSQL.
- No facilita ningún diseño entidad-relación ni tampoco ningún diagrama del modelo relacional.
- Hay que conocer los modelos más importantes.

Modelos y tablas.

- Odoo posibilita recuperar el nombre del modelo que define la información de un formulario y el nombre de los fields correspondientes a cada campo del formulario.
- El nombre de las clases y atributos siempre es en minúscula.
- Se utiliza el punto para definir jerarquía.
- En la base de datos, el punto es sustituido por un guion bajo.

Los módulos de odoo están organizados en ficheros. En las versiones actuales el archivo `__openerp__.py`, se denomina `__manifest__.py`



### Archivo `__init__.py`

- Importa archivos y directorios con código en python.
- Se crea en cada carpeta para importar los archivos Python.

### Archivo `__manifest__.py`

- Diccionario literal de python.
- Relaciona clave y valor.
- Determina los archivos XML que se analizarán durante la inicialización del servidor. (data).
- Determina las dependencias del módulo creado. (depends).
- Declara los metadatos adicionales.

### Modelos

- Todos los recursos de Odoo son modelos: facturas, socios, empleados, etc.
- Los metadatos también son modelos: menús, acciones, informes, etc.
- Los nombres de los modelos son jerárquicos, por ejemplo:
  - `account.transfer`: una transferencia de dinero.
  - `account.invoice`: una factura
  - `account.invoice.line`: una línea de factura.
- En general, la primera palabra es el nombre del módulo.
- El modelo es declarado en Python como subclase del modelo `models.model`.
- El ORM de Odoo es construido sobre PostgreSQL.
- Es peligroso escribir o leer directamente de la base de datos PostgreSQL.
- El ORM facilita la gestión de los datos.

### Fichero XML

- Inicialización y declaración de datos.
- Declaración de vistas.
- Declaración de informes.

## 2. CREANDO UN DIRECTORIO PARA DE MÓDULOS

Como ya vimos en la unidad 2, es importante tener un directorio para incluir y desarrollar módulos en Odoo. En este apartado comentaremos cómo poner en marcha este directorio tanto en una instalación manual como utilizando Docker.

### 2.1 Instalación manual

Si hemos hecho una instalación manual tal como se explicó en la unidad 02, una secuencia de comandos para conseguir crear este directorio en la carpeta “/var/lib/odoo/modules” es:

```
# Estando situado en /var/lib/odoo:
# Creamos directorio “modules”
mkdir modules
# Creamos un módulo llamado “prueba” con la utilidad “odoo scaffold”
odoo scaffold prueba ./modules
# Modificamos el path de los “addons”.
odoo --addons-path="/var/lib/odoo/modules,/usr/lib/python3/dist-packages/odoo/addons"
--save
# Lanzamos el servidor Odoo y actualizamos el módulo “prueba” en la bd “empresa”
odoo -u pruebas -d empresa
```

Con esta acción, lo que conseguimos es añadir nuestro directorio “modules” al PATH de Odoo en el archivo de configuración “**.odoorc**” propio del usuario Odoo. Con el último comando arrancamos Odoo actualizando este módulo (“prueba”) en la base de datos “empresa”.

Los módulos deben poder ser al menos leídos por el usuario “Odoo”, que es el que lanza el servicio.

- En un entorno de producción, los permisos de los módulos suelen configurarse de forma que solo el usuario “Odoo” pueda leerlos y ningún otro usuario pueda leer ni escribir.
- En un entorno de desarrollo, es recomendable que el propietario sea el propio “Odoo” y tenga permisos de escritura para hacer los cambios necesarios.

### 2.2 Instalación mediante Docker y Docker Compose

Si ponéis en marcha Odoo en modo desarrollo con Docker y Docker Compose tal como se sugería en la unidad 02, simplemente tendréis vuestro directorio de módulos listo para trabajar. Este directorio es el llamado “**volumesOdoo/addons**”, que tendréis mapeado en vuestra máquina anfitriona.

### 3. ENTORNO DE DESARROLLO

Existen diversos entornos de desarrollo que permiten desarrollar módulos para Odoo. Los más utilizados son “Visual Studio Code” y “PyCharm”. Además, es importante utilizar algunas herramientas adicionales. Recomendamos en el desarrollo utilizar un sistema de control de versiones “git” junto con plataformas como <https://github.com> o <https://gitlab.com>.

#### 3.1 Visual Studio Code

Recomendamos el uso de Visual Studio Code. Es muy potente y posee un gran ecosistema de plugins para ampliar su funcionalidad <https://code.visualstudio.com/>

Aquí algunos manuales libres de uso de Visual Studio Code en castellano:

<http://www.mclibre.org/consultar/informatica/lecciones/vsc-instalacion.html>

<http://www.mclibre.org/consultar/informatica/lecciones/vsc-personalizacion.html>

Para el desarrollo con Odoo, recomendamos los siguientes

- **Python:** <https://marketplace.visualstudio.com/items?itemName=ms-python.python>
- **Odoo-snippets:** <https://marketplace.visualstudio.com/items?itemName=jeffery9.odoo-snippets>

#### 3.2 Control de versiones usando Git + Visual Studio Code

Git es un sistema de control de versiones. Su uso en el entorno profesional esta muy extendido y es utilizado en todo tipo de desarrollos. Durante el curso, se utilizarán repositorios Git tanto para la entrega de prácticas como para facilitar el disponer de un repositorio con control de versiones.

Para instalar Git:

- Ubuntu:
  - `sudo apt-get update`
  - `sudo apt-get install git`
- Windows: <https://git-for-windows.github.io/>

Para facilitar la tarea del uso de Git es recomendable instalar alguna extensión o entorno que os facilite su uso. Para usar Git en Visual Studio Code recomendamos el siguiente plugin <https://marketplace.visualstudio.com/items?itemName=donjayamanne.git-extension-pack>.

Puedes encontrar más información de como usar Git con Visual Studio Code en:

- <https://code.visualstudio.com/docs/editor/versioncontrol>
- <http://www.mclibre.org/consultar/informatica/lecciones/vsc-git-repositorio.html>

### 3.3 PyCharm

Otro editor muy recomendado en Pycharm <https://www.jetbrains.com/es-es/pycharm/>  
Por defecto no reconoce los elementos del framework Odoo, pero sí los del lenguaje Python.  
Si queremos facilitar el desarrollo podemos instalar la extensión de Odoo Pycharm Templates <https://github.com/mohamedmagdy/odoo-pycharm-templates>.

### 3.4 Control de versiones usando Git + PyCharm

PyCharm también permite el uso de un sistema de control de versiones Git desde su entorno. Podéis encontrar información de como utilizar Git con Pycharm en:

- <https://www.jetbrains.com/help/pycharm/set-up-a-git-repository.html>
- <https://programmerclick.com/article/169045515/>

## 4. ACTIVANDO EL MODO DESARROLLADOR EN ODOO 17

En esta url de la guía de Odoo 17 se explican las distintas vías existentes para activar el modo desarrollador. Este modo nos permite depurar nuestros módulos Odoo gráficamente.

[https://www.odoo.com/documentation/17.0/es/applications/general/developer\\_mode.html](https://www.odoo.com/documentation/17.0/es/applications/general/developer_mode.html)

Cualquiera de las explicadas (interfaz Odoo, url, extensión navegador) es igualmente válida para este fin, aunque si realizáis un desarrollo exhaustivo es posible que la más cómoda sea la activación mediante una extensión del navegador.

Podéis encontrar la extensión para:

- Firefox: <https://addons.mozilla.org/es/firefox/addon/odoo-debug/>
- Chrome: [https://chrome.google.com/webstore/detail/odoo-debug/hmdmhilocobgohohpdpolmibjklfgkbi?hl=es\\_PR](https://chrome.google.com/webstore/detail/odoo-debug/hmdmhilocobgohohpdpolmibjklfgkbi?hl=es_PR)

**⚠ Atención:** para activar el modo desarrollador desde la interfaz Odoo, se debe haber instalado al menos un módulo (por ejemplo, Ventas).

## 5. NUESTRO PRIMER MÓDULO: “HOLA MUNDO”

Un módulo de Odoo sirve para extender las funcionalidades de este sistema ERP y puede tener



muchos propósitos distintos.

Para este ejemplo, vamos a crear el módulo más sencillo que permite Odoo 17. Un módulo que no hace absolutamente nada, salvo aparecer en la lista de módulos. La creación de este módulo tiene un fin completamente didáctico y nos ayudará a comprobar que nuestro sistema está configurado correctamente para poder detectar y utilizar los módulos que desarrollemos.

Para este primer “Hola mundo”, crearemos una carpeta “Ejemplo01-HolaMundo” dentro del directorio que hayamos configurado para colocar nuestros módulos.

Esta carpeta contendrá dos ficheros:

**a) Fichero “\_\_init\_\_.py”:**

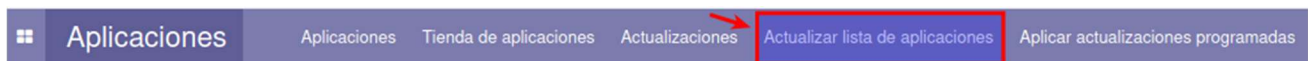
Este fichero estará vacío.

**b) Fichero “\_\_manifest\_\_.py”:**

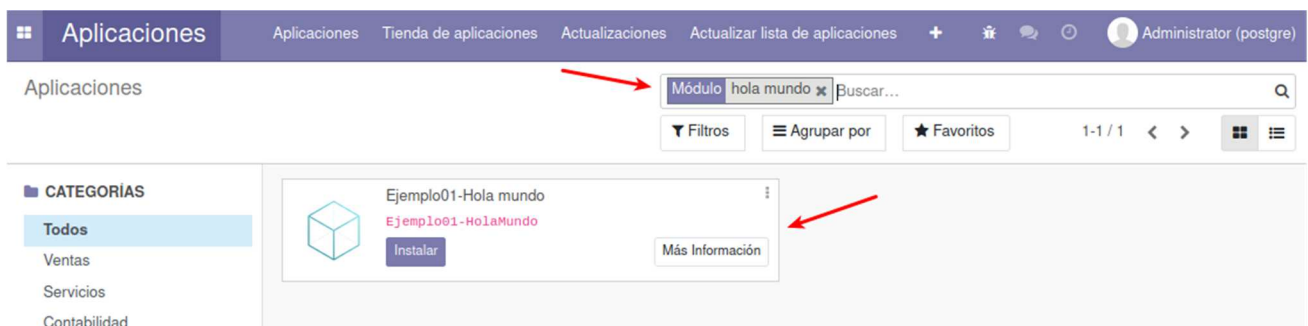
Este fichero contendrá el siguiente código:

```
# -*- coding: utf-8 -*-
{'name': 'Ejemplo01-Hola mundo'}
```

Una vez creada la estructura, con el “Modo Desarrollador” activado, podréis ir al listado de módulos y “Actualizar la lista de aplicaciones” tal como se observa en la imagen.



Tras ello, eliminando los filtros de búsqueda por defecto y buscando “hola mundo”, podremos encontrar nuestro módulo. Si todo ha funcionado correctamente, veremos algo similar a:



Ahora podremos instalar nuestro módulo para probarlo (aunque este ejemplo no hace nada).


## 6. CREANDO MÓDULOS EN ODOO

### 6.1 Creando módulos con “Odoo Scaffold”

Cuando creamos un módulo, digamos que se convierte en una “aplicación” dentro de “otra aplicación más grande” que es Odoo siendo esta aplicación prácticamente independiente del resto


de Odoo.

Los módulos de Odoo pueden tener muchas funcionalidades distintas. Vamos a comenzar realizando un tipo de módulo fácil de entender: un módulo que cree nuevos modelos de datos (ficheros maestros) y permita que se observen estos modelos a través de un nuevo menú.

 **Interesante:** otra posibilidad además de añadir modelos nuevos, es modificar modelos/vistas existentes para añadir funcionalidades a las que ya ofrece Odoo. Este tipo de módulos será tratado en la siguiente unidad didáctica mediante uso de herencia.

Una vez comprobado en puntos anteriores que tenemos listo todo lo necesario para trabajar con Odoo, explicaremos como desarrollar nuestro primer módulo con funcionalidad usando el comando “Odoo Scaffold” y basándonos en el ejemplo y la información descrita en:

[https://www.odoo.com/documentation/17.0/es/administration/odoo\\_sh/getting\\_started/first\\_module.html](https://www.odoo.com/documentation/17.0/es/administration/odoo_sh/getting_started/first_module.html)

 **Importante:** si habéis desplegado Odoo con Docker, para usar “odoo scaffold deberéis usarlo desde el interior del contenedor. Podéis hacerlo con un comando similar a “*docker exec -it IDCONEADOR /bin/bash*”. Si lo habéis hecho con Docker Compose, el comando equivalente podría ser “*docker-compose exec web /bin/bash*”.

Una vez todo listo, procedemos a crear nuestro módulo con una estructura base para trabajar usando “odoo scaffold” con un comando similar a:

```
odoo scaffold lista_tareas /mnt/extra-addons/
```

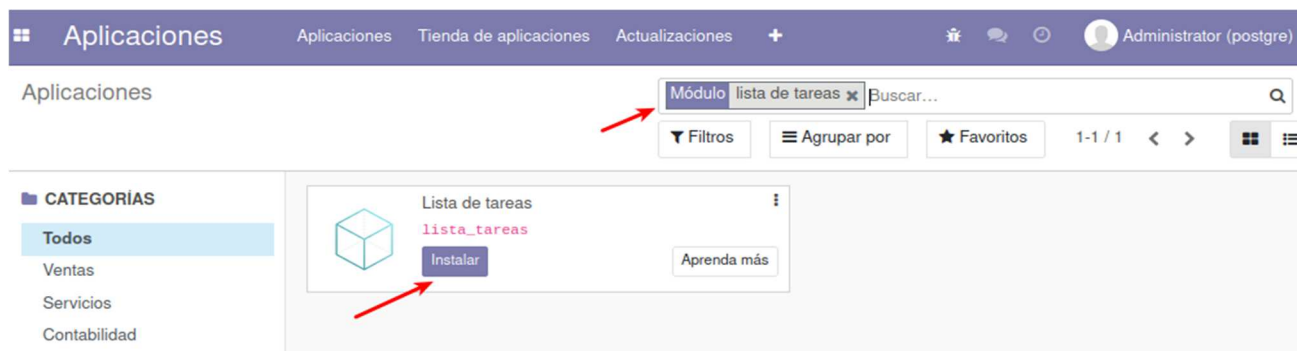
 **Atención:** el nombre del módulo no debe comenzar por número ni poseer el carácter “-”.

Si todo ha ido bien, dentro de la carpeta “/mnt/extra-addons” (o donde la hayáis montado si es un contenedor Docker), se habrá creado una carpeta “lista\_tareas”.

Si estáis dentro de un contenedor Docker, es recomendable darle permisos completos para poder editar fácilmente fuera del contenedor. Podéis hacerlo desde dentro del contenedor con:

```
chmod 777 -R /mnt/extra-addons/lista_tareas
```

Si actualizamos el listado de módulos, quitamos filtros y buscamos “lista tareas”, podremos acceder a él de forma similar a esta imagen:



Si queremos utilizar la estructura de ficheros que nos creará “odoo scaffold” sin utilizar este comando, la podemos descargar desde la siguiente dirección:

<https://www.odoo.com/documentation/17.0/es/downloads>

El módulo creado contiene un código de ejemplo, pero por defecto todo ese código está comentado. Si queremos habilitarlo, debemos descomentar el contenido de todos los ficheros creados.

## 6.2 Estructura de un módulo Odoo

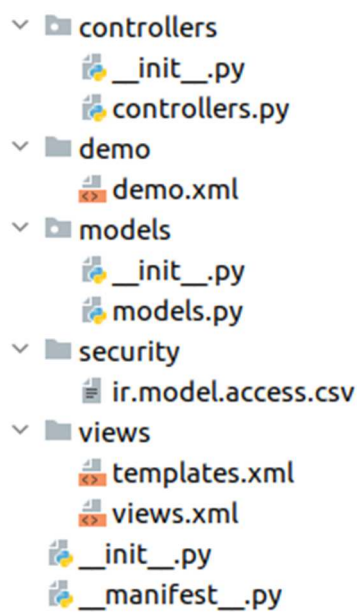
Como en cualquier framework, los directorios donde se programan módulos tienen unos ficheros con unos nombres y extensiones determinados.

En el caso de Odoo, todo empieza con un fichero Python llamado “**\_\_manifest\_\_.py**” que contiene la información necesaria para interpretar todos los ficheros que contiene el directorio. Esta información está almacenada usando una estructura diccionario de Python.

Además, como en cualquier paquete de Python, el directorio contiene un fichero “**\_\_init\_\_.py**”. Este tiene el nombre de los ficheros Python o directorios que contienen la lógica del módulo.

Los subdirectorios con ficheros Python de la estructura creada con “odoo scaffold” también tendrán su propio fichero “**\_\_init\_\_.py**”.

Internamente, la carpeta creada con “odoo scaffold” tiene el siguiente contenido:



Vamos a realizar una breve explicación de cada uno de los ficheros generados:

- “**models/models.py**”: define un ejemplo del modelo de datos y sus campos.
- “**views/views.xml**”: describe las vistas de nuestro módulo (formulario, árbol, menús, etc.).
- “**demo/demo.xml**”: incluye datos “demo” para el ejemplo propuesto de modelo.
- “**controllers/controllers.py**”: contiene un ejemplo de controlador de rutas, implementando algunas rutas.
- “**views/templates.xml**”: contiene dos ejemplos de vistas “qweb” usado por el controlador de rutas.
- “**\_\_manifest\_\_.py**”: es el manifiesto del módulo. Incluye información como el título, descripción, así como ficheros a cargar. En el ejemplo se debe descomentar la línea que contiene la lista de control de acceso en el fichero ‘**security/ir.model.access.csv**’.

### 6.3 Módulos en producción

En este apartado comentamos cómo funcionan los módulos cuando estamos trabajando en producción. Cuando instalamos o actualizamos un módulo de Odoo, el manifest indica al actualizador dónde están los ficheros de datos y otros parámetros para que actualice la base de datos.

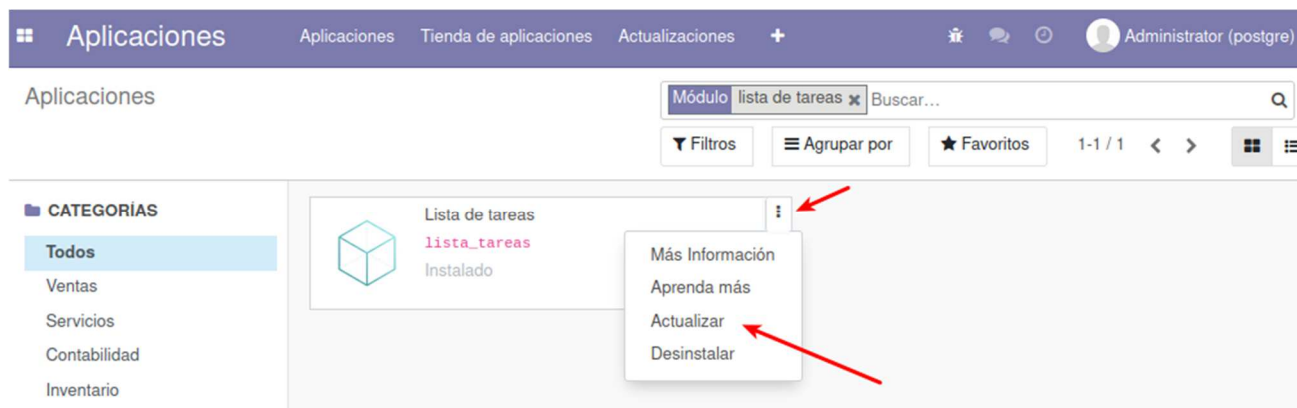
Odoo tiene una forma de funcionar “data-driven” (dirigido por datos) y cuando instalamos un módulo, las vistas, datos, etc. (los .xml) se almacenan en la base de datos. La actualización de la base de datos con datos, vistas, etc. sólo se hace **al instalar o actualizar el módulo**. Así que, aunque cambiemos una vista en un fichero XML, no se verá el cambio si no actualizamos el módulo Odoo (ni siquiera reiniciando el servicio Odoo).

Los ficheros Python de un módulo son cargados de nuevo cada vez que se inicia el servicio Odoo. Por lo que, si cambiamos algo en ellos, tenemos dos opciones para observar los cambios:

recargar el módulo o reiniciar el servicio Odoo (si necesidad de actualizar el módulo).

⚠ **Atención:** si estáis usando “Docker” podéis reiniciar el servicio simplemente reiniciando el contenedor “*docker restart IDCONTENEDOR*”. En nuestro ejemplo de “Docker Compose” sería “*docker-compose restart web*”.

Aquí una imagen donde se ve cómo actualizar un módulo ya instalado



## 6.4 Módulos en desarrollo

Si estamos trabajando en modo desarrollo (opción “`--dev=all`” al lanzar Odoo), el servidor Odoo leerá cada vez las vistas, datos y código Python directamente de los ficheros. Por lo tanto, podremos observar cambios de vistas, datos o código sin reiniciar el servicio ni actualizar el módulo.

⚠ **Atención:** esta práctica es muy cómoda para desarrollar, pero no se usa en entornos de producción por motivos tanto de seguridad como de rendimiento.

## 7. EJEMPLO DE MÓDULO “LISTA DE TAREAS”

Para poner en marcha nuestro primer módulo funcional, vamos a utilizar un ejemplo comentado donde crearemos una sencilla “Lista de tareas”.

Este módulo tendrá una estructura similar a esta:

Listado de tareas			
Opciones Lista Tareas		Administrator (postgre)	
Listado de tareas pendientes		<input type="text" value="Buscar..."/>	
<input type="button" value="Crear"/>		<input type="button" value="Filtros"/> <input type="button" value="Agrupar por"/> <input type="button" value="★ Favoritos"/> 1-2 / 2 < >	
<input type="checkbox"/> Tarea	Prioridad	Urgente	Realizada
<input type="checkbox"/> Hacer la compra	11	<input checked="" type="checkbox"/>	<input type="checkbox"/>
<input type="checkbox"/> Arreglar mi ordenador.	5	<input type="checkbox"/>	<input type="checkbox"/>

Y nos permitirá crear tareas con una prioridad asociada. También podremos marcar si la tarea está realizada o no. El campo “urgente” será un campo calculado (es decir, no se podrá editar a mano) que estará marcado si la prioridad es mayor que 10.

A continuación, desglosamos el código comentado de esta aplicación:

#### Fichero “\_\_manifest\_\_.py”:

```
# -*- coding: utf-8 -*-
{
    'name': "Lista de tareas",

    'summary': """
Sencilla Lista de tareas""",

    'description': """
Sencilla lista de tareas utilizadas para crear un nuevo módulo con un nuevo
modelo de datos
""",

    #Indicamos que es una aplicación
    'application': True,

    # En la siguiente URL se indica qué categorías pueden usarse
    #
https://github.com/odoo/odoo/blob/17.0/odoo/addons/base/data/ir\_module\_category\_data.x
    ml

    # Vamos a utilizar la categoría Productivity
    'category': 'Productivity',
    'version': '0.1',

    # Indicamos lista de módulos necesarios para que este funcione correctamente
    # En este ejemplo solo depende del módulo "base"
    'depends': ['base'],

    # Esto siempre se carga
```

```

    'data': [
        #Este primero indica la politica de acceso del módulo
        'security/ir.model.access.csv',
        #Cargamos las vistas y las plantillas
        'views/views.xml',
    ]
}

```

### Fichero “models.py”:

```

# -*- coding: utf-8 -*-

from odoo import models, fields, api

#Definimos el modelo de datos
class lista_tareas(models.Model):
    #Nombre y descripcion del modelo de datos
    _name = 'lista_tareas.lista_tareas'
    _description = 'lista_tareas.lista_tareas'

    #Elementos de cada fila del modelo de datos
    #Los tipos de datos a usar en el ORM son
    #
    https://www.odoo.com/documentation/17.0/developer/reference/addons/orm.html#fields

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

    #Este computo depende de la variable prioridad
    @api.depends('prioridad')
    #Funcion para calcular el valor de urgente
    def _value_urgente(self):
        #Para cada registro
        for record in self:
            #Si la prioridad es mayor que 10, se considera urgente, en otro caso no
            Lo es

            if record.prioridad>10:
                record.urgente = True
            else:
                record.urgente = False

```

### Fichero “views.xml”:

```

<odoo>
  <data>

```

```

<!-- explicit list view definition -->
<!-- Definimos como es la vista explicita de la listas-->
<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">
    <tree>
      <field name="tarea"/>
      <field name="prioridad"/>
      <field name="urgente"/>
      <field name="realizada"/>
    </tree>
  </field>
</record>
<!-- actions opening views on models -->
<!-- Acciones al abrir las vistas en los modelos
https://www.odoo.com/documentation/17.0/developer/reference/addons/actions.html
-->
<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>

<!-- 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"/>

</data>
</odoo>

```

## 8. BIBLIOGRAFÍA

- Sistemas de Gestión Empresarial IOC:  
[https://ioc.xtec.cat/materials/FP/Materials/2252\\_DAM/DAM\\_2252\\_M10/web/html/index.html](https://ioc.xtec.cat/materials/FP/Materials/2252_DAM/DAM_2252_M10/web/html/index.html)
- Wikipedia:  
[https://es.wikipedia.org/wiki/Sistema\\_de\\_planificaci%C3%B3n\\_de\\_recursos\\_empresales](https://es.wikipedia.org/wiki/Sistema_de_planificaci%C3%B3n_de_recursos_empresales)
- Documentación de Odoo:



<https://www.odoo.com/documentation/master/reference/http.html>