

SISTEMA DE GESTIÓN EMPRESARIAL

TÉCNICO EN DESARROLLO DE APLICACIONES MULTIPLATAFORMA

Creación de un módulo en Odoo

scaffold

Herencia

Log de Odoo

Errores en Odoo

Vamos a aprender a crear un módulo desde cero, en el cual, tendremos que implementar todos sus objetos (modelos, vistas, etc.), desde el inicio.

Veremos la **herencia de modelos y vistas ya existentes**.

Y hablaremos del **log de Odoo y los errores**, y cómo PyCharm nos ayudará a corregirlos.

## Creación de un módulo desde cero

En el presente punto describiremos los pasos necesarios para crear un módulo desde cero, que consistirán en:

### 1. Creación del módulo

- Crear un repositorio para almacenar los nuevos módulos.
- Agregar los **nuevos módulos** al `addon_path`.
- Crear el **esqueleto/plantilla** del módulo (`scaffold`).

### 2. Explotación del módulo

- Configurar e implementar el nuevo módulo.
- Instalar el nuevo módulo en Odoo y verificar su correcto funcionamiento.

#### Crear un repositorio para los módulos:

Es recomendable el uso de un repositorio para almacenar todos nuestros **módulos personalizados**. A este repositorio le llamaremos, por ejemplo, “`mismodulos`” y lo crearemos dentro de:

“C:\Program Files (x86)\Odoo 13.0\server\odoo”.

Ahí será donde crearemos las **distintas carpetas para los distintos módulos**.

#### Agregar los nuevos módulos al `addons_path`:

Una vez creado el repositorio, **deberemos** agregar la ruta de éste en el `addons_path` del fichero de configuración “`odoo.conf`”, cada uno de ellos **separados por ‘,’ (coma)**. Haremos lo mismo con los futuros módulos que creemos en este repositorio.

El **comando `scaffold`**: Este comando sirve para crear un **esqueleto/plantilla de módulo** con la **estructura** básica de un módulo de Odoo, donde aparecerá **código de ejemplo** comentado como ayuda. Su **uso no es obligatorio**, pero **facilita** la creación de nuevo módulos.

Hay varias **formas de ejecutar el `scaffold` desde el terminal**, pero en nuestro caso haremos uso de la siguiente:

```
<ruta donde se encuentra Python.exe> +  
<ruta donde se encuentra odoo-bin> +  
scaffold <nombre que tendrá el módulo> +  
ruta donde se almacenará el módulo.
```

Las rutas entre comillas dobles y el nombre del módulo sin comillas.

## Practicando con scaffold

Práctica con el comando Scaffold en Odoo desde PyCharm y Terminal

Practicaremos con el comando scaffold. Lo haremos tanto desde la aplicación PyCharm como desde el terminal.

Por ejemplo, esta es nuestra ruta de los módulos personalizados, que estarían en Odoo, mis módulos, y si por ejemplo quisiera crear un **nuevo módulo** con el comando scaffold, bastará con escribir en la terminal, como hemos visto:

- tanto la ruta donde se encuentra el ejecutable de Python,
- como la ruta donde se encuentra odoo-bin,
- seguido de scaffold,
- el nombre que queremos que tenga el nuevo módulo,
- y finalmente en qué lugar queremos que se almacene el nuevo módulo.

Por ejemplo, si ejecutamos, al pasar un momento vemos como si refrescamos la estructura de carpetas, vemos que hemos dicho que se guarde en server Odoo, mis módulos, el módulo que le hemos llamado librería.

También podemos hacerlo desde el terminal de Windows con el mismo comando, es decir, abrimos el terminal, escribimos el mismo comando, le vamos a cambiar, en este caso le vamos a poner el nombre librería2, y al ejecutarlo vemos como el comando scaffold actualiza, crea el nuevo módulo, librería2.

Estos como hemos visto tienen el **esqueleto básico que Odoo** nos proporciona para crear un nuevo módulo. Estos se encuentran también aquí, se han creado en el directorio concreto, con su carpeta de control, los controladores, los modelos, las vistas y el manifiesto que podemos abrir por ejemplo desde aquí mismo. Viene con una información por defecto que nosotros deberemos de modificar y al igual que los modelos vienen también con código de demostración comentado que nos puede servir de ayuda para ver lo que necesitamos escribir en estos objetos.

```
Microsoft Windows [Versión 10.0.18363.1816]
(c) 2019 Microsoft Corporation. Todos los derechos reservados.

C:\Program Files (x86)\Odoo 13.0\server>"C:\Program Files (x86)\Odoo 13.0\python\python.exe" "C:\Program Files (x86)\Odoo 13.0\server\odoo-bin" scaffold libreria "
C:\Program Files (x86)\Odoo 13.0\server\odoo\mismodulos\libreria"
```

## Configuración y verificación del módulo

### Configurando e implementando el nuevo módulo:

Una vez creado el esqueleto del módulo, lo siguiente será **informar el archivo 'manifest.py'**, con campos como:

- 'name' con el nombre que queramos que tenga el **nuevo módulo** en la lista de aplicaciones.
- 'summary' con la **descripción corta** de lo que hará el módulo.
- 'description' con la descripción que queramos que aparezca en la **información del módulo**.
- 'author' y 'website' con el autor y la página web que queremos que aparezca en la información del módulo.
- 'category' con la categoría del módulo de entre las que podremos encontrar en la siguiente web:  
[https://github.com/odoo/odoo/blob/13.0/odoo/addons/base/data/ir\\_module\\_category\\_data.xml](https://github.com/odoo/odoo/blob/13.0/odoo/addons/base/data/ir_module_category_data.xml).
- 'version' donde iremos **incrementando** la versión según vayamos realizando modificaciones.
- 'depends' con los **módulos** (no los modelos) que **necesitaría** como requisito nuestro nuevo módulo para funcionar, (y según un test y chatgpt: pudiendo estar vacío, tener uno o varios).
- 'data' con los **datos y vistas** que se usarán en el módulo.
- 'demo' con los **datos** que se usarán cuando se esté ejecutando Odoo en modo **demonstración**.

Ya solo quedaría **desarrollar los modelos, vistas, controladores**, etc. que sería la parte más compleja.

### Verificando el funcionamiento del nuevo módulo en Odoo:

Para que nos aparezca el nuevo módulo en la lista de aplicaciones de Odoo, deberemos pulsar sobre **'actualizar lista de aplicaciones'**.

A continuación, buscaremos nuestro nuevo módulo, que lo hemos llamado 'librería'.

Desde ahí podremos instalarlo (lo ampliaremos más adelante). Si clicamos sobre **'Aprenda más'** el navegador abrirá una nueva pestaña con la **web que hayamos informado** en el manifest. Si por el contrario pulsamos sobre **'información del módulo'**, se nos mostrará la información que hemos modificado en el **manifest**.

## Creación de un módulo con herencias

Hay veces que queremos realizar modificaciones sobre aplicaciones ya existentes, ya sean estándares o no. Para ello, Odoo ofrece las herencias, que son una característica que nos permite realizar **modificaciones sobre objetos** sin necesidad de realizar las implementaciones sobre ellos. Es decir, al heredar se crea una especie de capa sobre el objeto original con las modificaciones que se quiera implementar sobre éste, pero **sin modificar el original** y con la posibilidad de **instalarlo y desinstalarlo cuando se requiera**. O lo que es lo mismo, en lugar de modificar un módulo existente, creamos un **nuevo módulo para realizar las modificaciones oportunas** (agregar o quitar funcionalidad) sobre el original. Tal y como vimos, la herencia puede afectar a cada componente del MVC (modelo-vista-controlador).

Según un test:

Con la herencia, en lugar de modificar un módulo existente, creamos un **nuevo módulo** para realizar las **modificaciones oportunas** (agregar o quitar funcionalidad) **sobre el original**.

Antes de crear nuevos campos programando con herencia, vamos a **buscar** con la ayuda de PyCharm un **modelo concreto entre todos los módulos** de Odoo, puesto que hay veces que queremos copiar o modificar un modelo concreto, pero no sabemos dónde se ubica. Esto nos será de mucha utilidad a la hora de crear herencias.

### Ver la ubicación de un modelo

Vamos a ver un truco para buscar dónde se encuentra ubicado el modelo concreto que estamos tratando, en qué módulo, por ejemplo. En este caso queremos hacer alguna modificación o copiar cualquier cosa de esta ventana.

Sabemos que el modelo es SALEORDER. Entonces, para buscar dentro de qué módulo está, porque hay veces que es complicado encontrar el código, podemos hacer uso de PyCharm de la siguiente forma:

Sabemos que va a estar dentro de una carpeta de SALE, de ventas, pero hay muchas.

Entonces podemos pulsar **botón derecho sobre la carpeta de addons**, que son los módulos, y buscar en el **path**. A continuación **marcaremos los archivos Python**, porque lo que necesitamos es buscar el lugar **dónde** se está **definiendo el modelo**.

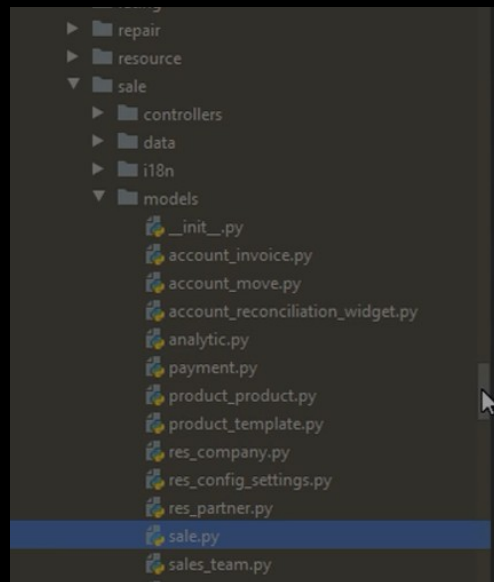
El modelo se define de la siguiente forma, con un

```
_name = SALEORDER
```

Si os fijáis solo encuentra una coincidencia que es esta, porque el modelo solo se define una única vez. Sin embargo, si en vez de 'SALEORDER' ponemos 'INERIT', aparecen muchas más, porque esto es **todas las herencias** de ese modelo.

Por ejemplo, si volvemos a la definición del modelo y pulsamos aquí doble clic, ya nos lleva al modelo en concreto donde se está declarando. Si pulsamos botón derecho File Path, nos muestra la ruta donde se encuentra concretamente el modelo. Es decir, está en la carpeta 'SERVER > ODOO > ADDONS', en el módulo de 'SALE', la carpeta 'modelos' y 'SALE PATH'.

Es decir, la ruta sería la siguiente: 'SALE', en 'modelos', y aquí encontraríamos el modelo de 'SALE'. Que es el que, donde se está definiendo, el modelo que nos interesa.



Una vez que sepamos como encontrar el modelo del que vamos a heredar, empezaremos a trabajar la herencia con un ejemplo: Concretamente, en el **módulo de Ventas**, agregaremos un nuevo campo **debajo de 'Plazos de pago'** donde podremos informar el **'modo de pago'** de entre los indicados en un desplegable (metálico, transferencia bancaria o tarjeta).

En primer lugar, realizaremos los pasos que hemos visto en este tema hasta ahora, es decir,

- realizaremos un 'scaffold' para crear el **esqueleto del nuevo módulo**, al que llamaremos **'modos de pago'**,
- **borraremos** los objetos que en principio **no necesitaremos** (como, por ejemplo, las carpetas de 'controllers' y 'demo'),
- **quitaremos del archivo '\_\_init.py'** las carpetas que **hayamos eliminado** previamente
- y, finalmente, informaremos el archivo **manifest**.

Ahora, ya estaremos preparados para **implementar el modelo** y la vista con herencias en lugar de implementarlos desde cero.

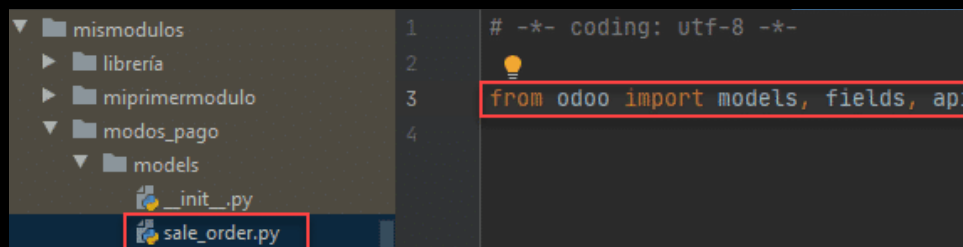
## Herencia del modelo

Queremos agregar en el módulo de Ventas un nuevo campo debajo de 'Plazos de pago' donde podremos informar el 'modo de pago' de entre los indicados en un desplegable (metálico, transferencia bancaria o tarjeta).

Empezaremos creando el modelo, pero debemos hacerlo sin tocar el código estándar de Odoo, es decir haciendo uso de herencias.

Tal y como hemos aprendido, ya sabemos que tendremos que implementar una **herencia de 'sales.order'**. Para ello, lo primero que haremos como **buena praxis** será **renombrar el modelo de la carpeta 'model'** con el nombre del **modelo del que heredaremos** (en este caso sale\_order), lo reemplazaremos en el '\_\_init\_\_.py' de la **carpeta 'models'**, y limpiaremos el código que se nos crea de plantilla del scaffold, **dejando** solo la línea con la importación de las **librerías básicas**.

### Limpiando el código



A continuación, crearemos la **clase 'SaleOrder'** como en la declaración del objeto original, solo que en este caso solo **informaremos** el **atributo '\_inherit'**, para indicar que este modelo **es heredado** del modelo 'sale.order'.

Finalmente, agregaremos el **campo 'modo de pago'** de **tipo 'selection'** con las tuplas:

- **Clave:** Lo que se guardará en base de datos.
- **Valor:** Lo que se mostrará en el desplegable.

Heredando el modelo y creándole el nuevo campo



```
from odoo import models, fields, api

class SaleOrder(models.Model):
    _inherit = 'sale.order'

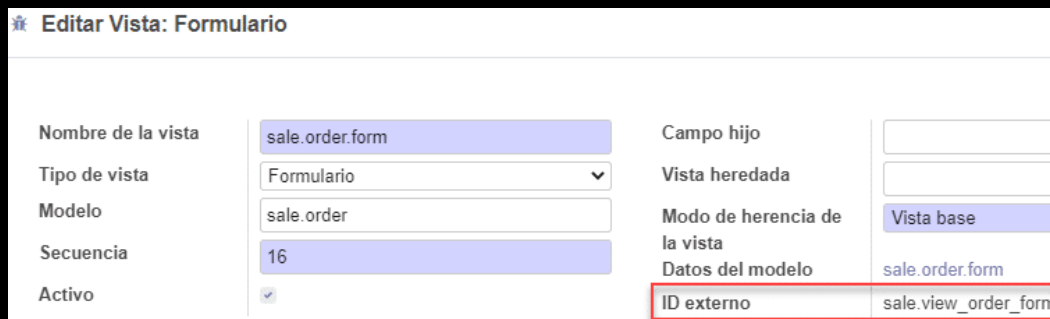
    modo_pago = fields.Selection(
        string="Modo de pago",
        selection=[
            ('efectivo', "Efectivo"),
            ('tarjeta', "Tarjeta"),
            ('transferencia', "Transferencia bancaria")
        ]
    )
```

Con esto ya tendríamos implementado el **modelo del módulo** con el **nuevo campo desplegable**, ahora nos quedará implementar la vista (para poder visualizar el campo) e instalar y probar el módulo.

## Heredar la vista

En primer lugar, tendremos que buscar la vista de la que queremos heredar. Para ello, nos posicionaremos en Odoo en la ventana de Ventas **en la que queremos agregar el campo**, pulsaremos sobre la opción '**Editar Vista: Formulario**' del '*Open Developer Tool*' (el icono del insecto), y el campo '*ID externo*' nos indicará la vista de la que tendremos que heredar:

ID externo de la vista de la que queremos heredar



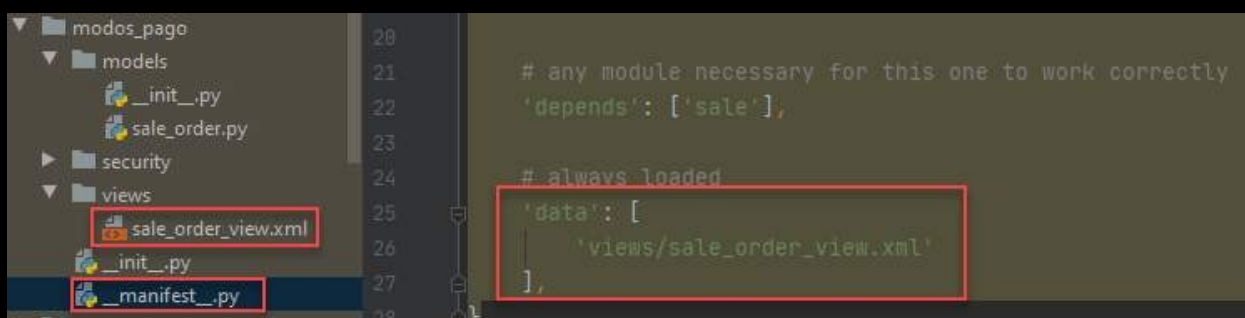
Nombre de la vista	sale.order.form	Campo hijo	
Tipo de vista	Formulario	Vista heredada	
Modelo	sale.order	Modo de herencia de la vista	Vista base
Secuencia	16	Datos del modelo	sale.order.form
Activo	<input checked="" type="checkbox"/>	ID externo	sale.view_order_form

A continuación, en la carpeta '*views*' de nuestro módulo **eliminaremos** las vistas **ejemplo** que crea la plantilla y **crearemos una nueva** que se llamará '*sale\_order\_view.xml*'.

Seguidamente, agregamos esta **vista en el 'manifest'** en la sección '*data*', con el formato:

**'data': ['views/sale\_order\_view.xml']:**

Agregando la vista en el manifest



Ya solo nos quedará implementar el código de la vista en la que **heredaremos de la vista sale.view\_order\_form**.

En primer lugar, crearemos una **estructura <data>** para un archivo XML y en su interior haremos la **herencia** de la vista. En la herencia informaremos:

- el 'name' de la vista,
- el 'model' del que heredamos,
- y el 'inherit\_id' con el ID externo (que vimos en odoo, en "editar vista formulario" del *Open Developer Tool*).

Finalmente, en el interior de la etiqueta 'arch' le indicaremos que el campo '**modo\_pago**' irá **después** del campo '**payment\_term\_id**' (name="payment\_term\_id") y (position="after").

Heredando la vista y posicionando el nuevo campo

```
<?xml version="1.0" encoding="utf-8"?>
<odoo>
  <data>
    <!-- Inherit Form View to Modify it -->
    <record id="view_order_form" model="ir.ui.view">
      <field name="name">sale.order.form.modo_pago</field>
      <field name="model">sale.order</field>
      <field name="inherit_id" ref="sale.view_order_form"/>
      <field name="arch" type="xml">
        <field name="payment_term_id" position="after">
          <field name="modo_pago"/>
        </field>
      </field>
    </record>
  </data>
</odoo>
```

## Instalación del módulo

Antes de instalar el módulo, vamos a ver **cuántos campos** tiene el **módulo de Ventas**.

Para ello, en Odoo:

1. navegamos al módulo de **ajustes**,
2. abrimos **técnico**/modelos,
3. buscamos el **módulo 'sale.order'**
4. y vemos que dispone de 89 campos.

A continuación, instalaremos el nuestro y comprobaremos que ese número se habrá incrementado en 1 campo.

Para instalar el módulo, tal y como explicamos en el punto 3 del presente tema, bastará con 'actualizar lista de aplicaciones', buscar nuestro nuevo módulo 'Modos de pago' e **instalarlo**.

Tras su instalación, si volvemos a comprobar el número de campos que tiene el módulo de Ventas, ahora serán 90.

Entre los que se encuentra el nuestro 'modo\_pago'.

Si ahora abrimos el módulo de Ventas, comprobaremos que el campo se ha agregado correctamente en el lugar que queríamos.

## La importancia de la herencia

Tal y como hemos visto, la herencia es algo muy importante para el programador de Odoo puesto que nos facilitará mucho el trabajo. Nos permitirá ir agregando o quitando piezas a un puzzle que puede ser todo lo grande o pequeño que queramos. Además, con la ventaja de que dichas piezas se pueden poner y quitar cuando sea necesario sin interferir el resto del puzzle.

La **funcionalidad base** del código estándar de Odoo **siempre estará ahí**, y podremos ampliarla haciendo uso de herencias.

## El log de Odoo

Por defecto Odoo muestra todo el registro de log en el 'stdout' (**standar out** o salida estándar, que en PyCharm es el menú que aparece en la sección 'run'), pero hay varias opciones para redirigir el registro a otros destinos y personalizar la **cantidad de salida de información** (todas **configurables** en el registro de **configuración odoo.conf**).

A continuación, comentaremos los más útiles de las descritas en la web de Odoo, en la sección logging:

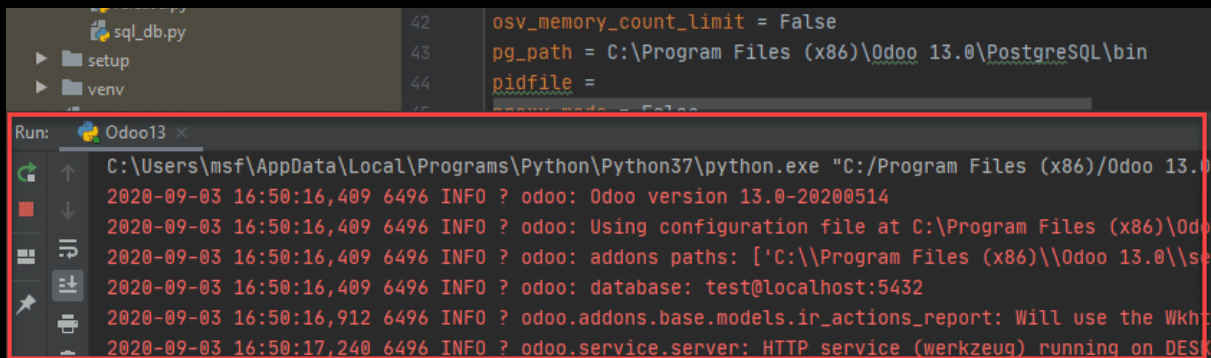
**logfile** <archivo>:

Envía la salida de registro **al archivo especificado** en lugar de stdout. Por ejemplo, por defecto se informa en el directorio "C:\Program Files (x86)\Odoo 13.0\server\odoo.log":

Si está **informado el parámetro logfile**, ese archivo **almacenará todos los logs** que se produzcan en Odoo en el nivel que **se indique en el log\_handler**.

Si no está informado aparecerá en el stdout en PyCharm:

### Stdout de PyCharm

The image shows a screenshot of the PyCharm IDE. At the top, there's a code editor with Python code for Odoo configuration, including 'osv\_memory\_count\_limit = False', 'pg\_path = C:\Program Files (x86)\Odoo 13.0\PostgreSQL\bin', and 'pidfile ='. Below the code editor is the 'Run' console, which is titled 'Odoo13'. The console shows the execution of a Python command: 'C:\Users\msf\AppData\Local\Programs\Python\Python37\python.exe "C:/Program Files (x86)/Odoo 13.0...'. The output of the command is displayed in the console, showing various log messages from Odoo, including version information, configuration file location, addons paths, database connection, and service status. The log messages are timestamped and include log levels like INFO and DEBUG.

**log\_handler** <:NIVEL>:

habilita el **log en el NIVEL proporcionado**, por ejemplo:

:DEBUG habilitará todos los mensajes de registro **en o por encima del nivel DEBUG**. Los niveles son:

- critical,
- error,
- warning,
- info,
- debug,
- notset.

Además de en el **archivo de configuración**, se puede tratar el manejador de error en cada ejecución **diferenciando objetos**:

```
$ odoo-bin --log-handler :DEBUG --log-handler odoo.models:CRITICAL --log-handler  
odoo.fields:WARNING
```

## Analizando el log al ejecutar Odoo desde PyCharm

Al ejecutar Odoo desde PyCharm aparece una información en el log.

¿Qué quiere decir esta información? Por defecto el manejador se encuentra en :INFO ¿mostrará la misma información si lo ponemos en :DEBUG?

Así es, al ejecutar el intérprete de Odoo en PyCharm:

Ejecutando el intérprete



Nos aparece el siguiente log:

Log en modo :INFO

```
2020-09-03 16:50:16,409 6496 INFO ? odoo: Odoo version 13.0-20200514
2020-09-03 16:50:16,409 6496 INFO ? odoo: Using configuration file at C:\Program Files (x86)\Odoo 13.0\server\odoo.conf
2020-09-03 16:50:16,409 6496 INFO ? odoo: addons paths: ['C:\\Program Files (x86)\\Odoo 13.0\\server\\odoo\\addons']
2020-09-03 16:50:16,409 6496 INFO ? odoo: database: test@localhost:5432
2020-09-03 16:50:16,912 6496 INFO ? odoo.addons.base.models.ir_actions_report: Will use the Wkhtmltopdf library
2020-09-03 16:50:17,240 6496 INFO ? odoo.service.server: HTTP service (werkzeug) running on DESKTOP-4HNEI
```

Con la siguiente información:

- La **versión de Odoo**.
- La **ruta del archivo de configuración** (odoo.conf) que se está usando.
- La/s ruta/s de los **addons paths**, que están informadas en el archivo de configuración previo.
- La **base de datos** que se está usando.
- Se indica que se usará el **programa Wkhtmltopdf**.
- Se indica que donde está corriendo el **servicio HTTP**.

Si cambiamos el manejador a: DEBUG y volvemos a arrancar el intérprete, aparece más información:

Log en modo :DEBUG

```
2020-09-03 17:30:28,094 8056 DEBUG ? odoo.netsvc: logger level set: "odoo.http.rpc.request:INFO"
2020-09-03 17:30:28,094 8056 DEBUG ? odoo.netsvc: logger level set: "odoo.http.rpc.response:INFO"
2020-09-03 17:30:28,094 8056 DEBUG ? odoo.netsvc: logger level set: ":INFO"
2020-09-03 17:30:28,094 8056 DEBUG ? odoo.netsvc: logger level set: ":DEBUG"
2020-09-03 17:30:28,094 8056 INFO ? odoo: Odoo version 13.0-20200514
2020-09-03 17:30:28,094 8056 INFO ? odoo: Using configuration file at C:\Program Files (x86)\Odoo 13.0
2020-09-03 17:30:28,094 8056 INFO ? odoo: addons paths: ['C:\\Program Files (x86)\\Odoo 13.0\\server\\
2020-09-03 17:30:28,094 8056 INFO ? odoo: database: test@localhost:5432
2020-09-03 17:30:28,717 8056 INFO ? odoo.addons.base.models.ir_actions_report: Will use the Wkhtmltopd
2020-09-03 17:30:29,060 8056 DEBUG ? odoo.service.server: Setting signal handlers
2020-09-03 17:30:29,060 8056 DEBUG ? odoo.service.server: cron0 started!
2020-09-03 17:30:29,060 8056 DEBUG ? odoo.service.server: cron1 started!
2020-09-03 17:30:29,075 8056 INFO ? odoo.service.server: HTTP service (werkzeug) running on DESKTOP-4H
2020-09-03 17:31:29,075 8056 DEBUG ? odoo.service.server: cron0 polling for jobs
2020-09-03 17:31:30,079 8056 DEBUG ? odoo.service.server: cron1 polling for jobs
```



## Errores o Dumps

Puede darse el caso de que al programar cometamos **errores sintácticos**. En esos casos, PyCharm resaltará dichos errores de código en tiempo real, es decir, los mostrará a medida que se está escribiendo. Por ejemplo, si se nos olvidase cerrar una etiqueta en un archivo XML que estemos editando, PyCharm nos mostrará con:

- una **exclamación roja en la parte superior derecha** que ese objeto tiene errores
- y además nos indicará con una **línea roja** también, **dónde y cuál es el error** concreto:

### Errores de código en tiempo real



Una vez solucionado el error, la exclamación roja de la parte superior derecha pasará a ser un **tick verde**, que nos indicará que el **objeto actual no tiene errores** de sintaxis.

## Código sin errores de sintaxis

```
-- Inherit Form View to Modify it -->
record id="view_order_form" model="ir.ui.view">
  <field name="name">sale.order.form.modelo.pago</field>
  <field name="model">sale.order</field>
  <field name="inherit_id" ref="sale.view_order_form"/>
  <field name="arch" type="xml">
    <field name="payment_term_id" position="after">
      <field name="modelo_pago"/>
    </field>
  </field>
</field>
```

También pueden darse otros tipos de errores, como los **ortográficos**, que **no son capturados en tiempo real por PyCharm**. Por ejemplo, si en lugar de heredar en nuestro ejemplo anterior del ID externo de la vista padre 'sale.view\_order\_form' lo hiciésemos por error de 'sale.view\_order\_formulario' (que no existe), al intentar instalar el módulo será cuando Odoo nos muestre el error:

## Error en tiempo de ejecución

**Error de Odoo**

Se ha producido un error [Copiar el error al completo](#)

Utilice el botón "Copiar" para informar al servicio de asistencia del error.

[Ver detalles](#)

```
File "C:\Program Files (x86)\Odoo 13.0\server\odoo\tools\convert.py", line 663, in model_id_get
    return self.env['ir.model.data'].xmlid_to_res_model_res_id(id_str, raise_if_not_found=raise_if_not_found)
File "C:\Program Files (x86)\Odoo 13.0\server\odoo\addons\base\models\ir_model.py", line 1681, in xmlid_to_res_model_res_id
    return self.xmlid_lookup(xmlid)[1:3]
File "<decorator-gen-24>", line 2, in xmlid_lookup
File "C:\Program Files (x86)\Odoo 13.0\server\odoo\tools\cache.py", line 90, in lookup
    value = d[key] = self.method(*args, **kwargs)
File "C:\Program Files (x86)\Odoo 13.0\server\odoo\addons\base\models\ir_model.py", line 1670, in xmlid_lookup
    raise ValueError('External ID not found in the system: %s' % xmlid)
odoo.tools.convert.ParseError: "External ID not found in the system: sale.view_order_formulario" while parsing file:c:/program%20
<odoo>
<data>
  <!-- Inherit Form View to Modify it -->
  <record id="view_order_form" model="ir.ui.view">
```

Al analizar este error en tiempo de ejecución, podremos observar que nos está indicando que **no se ha encontrado el ID externo**. Esto nos indicará lo que deberemos solucionar para poder instalar el módulo de manera satisfactoria.

Hemos:

- Aprendido a crear un **módulo desde cero**.
- Creado un **repositorio** para los módulos personalizados y lo hemos **referenciado en el addon\_path**.
- Conocido la funcionalidad y cómo usar el comando **scaffold**.
- **Configurado, implementado, instalado y probado** el nuevo **módulo**.
- Hecho uso de las **herencias**.
- Configurado y analizado algunos **logs y errores** que ofrece Odoo.

Podemos ofrecer la posibilidad a nuestro cliente de crearle un nuevo módulo que el podrá instalar/desinstalar cuando desee. Este nuevo módulo será un **módulo heredado** del módulo estándar de Odoo y le agregará la **nueva funcionalidad** que el cliente requiere.

De esta forma, nosotros simplemente crearemos un **nuevo módulo** en el cual implementaremos la herencia **desde el módulo estándar** y agregaremos la **nueva funcionalidad**. Así, cuando el cliente actualice la lista de aplicaciones, le aparecerá el nuevo módulo, con el cual **podrá instalar y desinstalar** la funcionalidad cuando quiera.

[https://www.odoo.com/es\\_ES/](https://www.odoo.com/es_ES/)

<https://praxyaformaplus.com/>

<https://odooerpcloud.com/>