

SISTEMA DE GESTIÓN EMPRESARIAL

TÉCNICO EN DESARROLLO DE APLICACIONES MULTIPLATAFORMA

**Cómo funciona Odoo por dentro**

**Módulos**

**Modelos**

**Capa ORM**

**Vistas**

**Informes**

**Controladores**

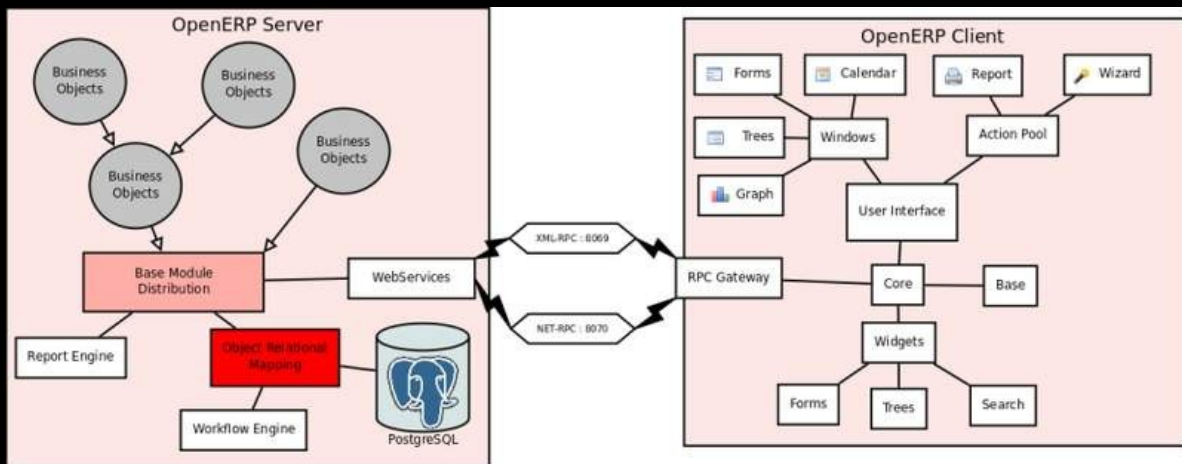
**Campos calculados**

**Configuración de PyCharm para Odoo**

## Principios del funcionamiento de Odoo

Odoo usa una **arquitectura** cliente/servidor en la que los **clientes son navegadores web que acceden al servidor Odoo a través de RPC** (Remote Procedure Call o llamada a procedimiento remoto). El cliente se comunica con el servidor en **XML-RPC**, un protocolo de llamada a procedimiento remoto que usa **XML para codificar los datos** y **http** como protocolo de transmisión de mensajes.

Estructura cliente/servidor de Odoo



La **lógica y la extensión** del negocio, generalmente, se realizan en el lado del **servidor**.

Para comprender cómo funciona Odoo realmente por dentro, deberemos conocer lo siguiente:

- Existe una **capa ORM** (Object Relational Mapping) entre los objetos Python y la base de datos PostgreSQL, de forma que el **programador** no efectuará el diseño de la base de datos, este **creará clases** a las que la capa **ORM mapeará** con el SGBD. Gracias al ORM, no es necesario programar con **consultas SQL**, ya que esta capa proporciona una **serie de métodos** que lo facilita. Ahora, en lugar de hablar de tablas, se hablará de **modelos** que son **mapeados por el ORM en tablas**. Un modelo no es una tabla, es un **objeto con campos funcionales, restricciones** y campos **relacionales**.
- Odoo usa una **arquitectura modelo-vista-controlador** (MVC):
  - **Modelo**: Clases diseñadas en **Python**.
  - **Vista**: Formularios, vistas, etc., definidos en **XML**.
  - **Controlador**: Reside en los **métodos** definidos en las **clases** que proporcionan la **lógica de negocio**.

Tanto las **extensiones** de servidor como las de cliente se empaquetan como **módulos** (módulo de compra, de ventas, CRM, facturación, etc.) que se **instalan y almacenan opcionalmente** en la base de datos.

Cada módulo de Odoo puede agregar una **lógica comercial nueva** o **alterar/ampliar** la lógica comercial existente en el sistema Odoo. “Todo en Odoo comienza y termina con módulos”.

## Estructura interna de los módulos

Los módulos de Odoo amplían o modifican partes de modelo-vista-controlador.

De este modo, **un módulo puede tener:**

- **Objetos de negocio:**

Son la parte del **modelo**, están **definidos en clases** de Python según una sintaxis propia del ORM de Odoo.

- **Vistas de objetos:**

Definición de **visualización de IU** de objetos de negocio.

- **Ficheros de datos:**

Son **archivos XML o CSV** que pueden definir **datos, vistas, configuraciones**, etc.

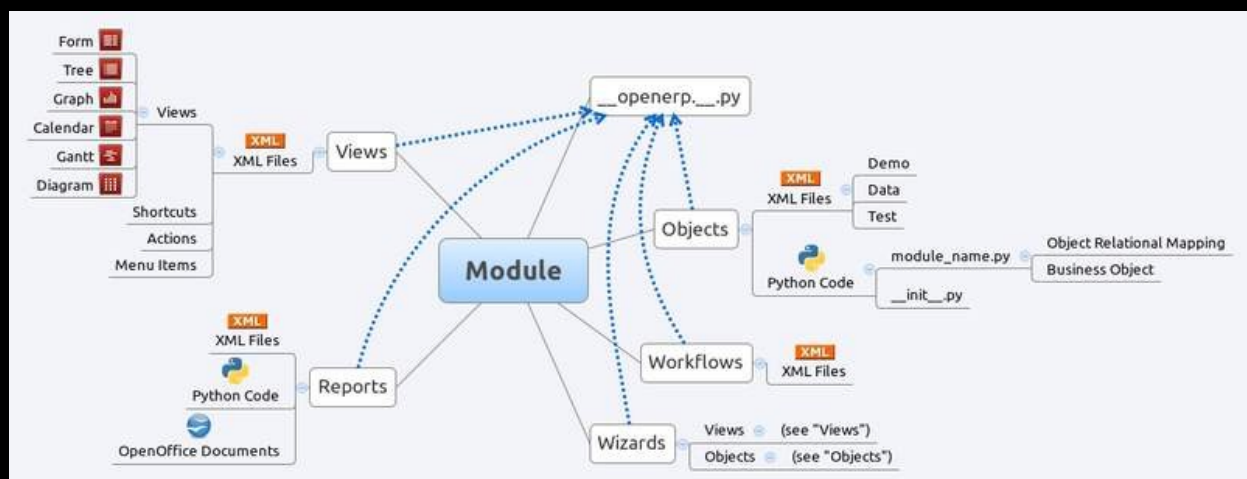
- **Controladores web:**

**Gestionan las solicitudes** de los navegadores web.

- **Datos estáticos:**

Como pueden ser **JavaScript, imágenes** u hojas de estilo en cascada (**CSS**) utilizados en la interface web.

Los módulos en Odoo



Los módulos requieren tener en su interior los siguientes archivos:

- `'__manifest__.py'`:

Sirve para declarar un paquete de Python **como un módulo** Odoo y para especificar **metadatos** del módulo. Además, ayuda a **mostrar** el módulo dentro de la **lista de aplicaciones** Odoo. Dentro de él, se especifica el **nombre** del módulo, la **versión**, una **descripción**, las **dependencias** con otros módulos, etc. Tienes más información aquí.

El archivo manifest

```
{
    'name': "A Module",
    'version': '1.0',
    'depends': ['base'],
    'author': "Author Name",
    'category': 'Category',
    'description': """
    Description text
    """,
    # data files always loaded at installation
    'data': [
        'views/mymodule_view.xml',
    ],
    # data files containing optionally loaded demonstration data
    'demo': [
        'demo/demo_data.xml',
    ],
}
```

- `'__init__.py'`:

En él, se implementan **instrucciones para importar archivos** Python.

Por ejemplo, si el módulo tuviese un solo archivo Python denominado “mimodulo.py”, el “init” tendría una sola línea de código ‘from . import mimodulo’.

Los módulos se encuentran en el directorio que se especifique en el ‘addons\_path’ del archivo de configuración de Odoo (odoo.conf), pudiendo ser más de un directorio, como veremos este vídeo.

## La estructura del módulo

En el presente vídeo, analizaremos de dónde se configuran los módulos con los que trabaja nuestro Odoo, y veremos un esqueleto básico de un módulo de Odoo. Como hemos comentado en la documentación, los módulos con los que va a trabajar nuestro Odoo se encuentran en el addon path del archivo de configuración de Odoo. Pueden ser varios, y en este caso, se están trayendo todos los módulos de esta ruta de addons, también de Odoo, mis módulos y de mis módulos, mi primer módulo.

Este es el archivo de configuración de Odoo, como ya sabemos, está dentro de esta ruta y un ejemplo de los módulos de un esqueleto de un módulo podría ser el siguiente. Este esqueleto lo hemos creado con el comando `scaffold` que más adelante comentaremos lo que es, pero adelantamos que crea un **esqueleto básico de un módulo** de Odoo. En este caso, le hemos puesto el nombre del módulo `ProductPrice2`, y como vemos, tiene nuestro archivo `__init__.py` que ahora mismo está vacío, y nuestro **manifiesto** que está informado, pero por defecto con un nombre, una descripción y demás. Esto lo tendremos que editar una vez que comencemos a crear este módulo.

Además, tenemos la `carpeta views` donde están nuestros módulos de vista que son los XML como hemos visto, los modelos, los controladores, etc.

## Los modelos y la capa ORM (Object Relational Mapping)

Odoo mapea sus objetos en una base de datos con la capa ORM, evitando así tener que realizar consultas SQL. De esta forma, la **capa ORM** de Odoo facilita unos **métodos** que se encargan del **mapeo** entre los **objetos Python** y las **tablas de PostgreSQL**, por ejemplo, para crear, modificar, eliminar y buscar registros en la base de datos.

Los **modelos** son una manera de **relacionar el programa con la base de datos**. Realmente, para comprender qué son, podríamos asemejarlos a las tablas de la base de datos en Odoo, aunque no lo sean realmente. Se crean como **clases de Python** que pueden **extenderse** de:

- La clase **models.Model** para modelos **persistentes** en la base de datos.
- La clase **models.TransientModel** para datos **temporales** almacenados en la base de datos, pero que se borran automáticamente cada cierto tiempo.
- La clase **models.AbstractModel** para **superclases abstractas** destinadas a ser compartidas por múltiples modelos que la heredarán.

Los modelos contendrán los campos y métodos útiles para manejar el ORM. Al crearlos, es necesario dar **valores a algunas variables**, como **\_name**, que es obligatoria y la que da nombre al modelo en Odoo. De esta forma, la definición mínima de un modelo sería la siguiente:

Definición mínima de un modelo

```
from odoo import models
class MinimalModel (models.Model):
    _name = 'test.model'
```

Los modelos son de **más alto nivel** que las consultas directas a **base de datos** y que las **clases y objetos** respecto a la **programación orientada a objetos**.

Unen en un **único concepto** las **estructuras** de datos, las **restricciones** de integridad y las opciones de **manipulación** de los datos.

Para ver los modelos existentes, se puede acceder a la base de datos PostgreSQL o mirar en **Configuración> Estructura de la base de datos> Modelos** dentro del modo desarrollador.

Fig. 7. Visualizando los modelos desde Odoo.

## Los campos (fields) del modelo

Los campos (*fields*) o “columnas” se usan para indicar **qué puede almacenar** el modelo y dónde. Se implementan como **atributos** en la clase y se **declaran como un constructor**:

Declarando un campo en el modelo

```
from odoo import models, fields

class LessMinimalModel (models.Model):
    _name = 'test.model'

    name = fields.Char()
```

Pueden ser de **tipos**:

- De **datos simples**: Como integer, boolean, date, char, etc.
- **Reservados**: El **sistema gestiona estos campos** y no debe escribirse en ellos, solo leerlos si fuese necesario. Por ejemplo: id, create\_date, create\_uid, etc.
- **Especiales**: Como **many2one**, **one2many**, **related**, etc.

Además, cuando se está creando un campo, se puede usar una serie de **atributos**, como **readonly**, **required**, etc.

Para **acceder a los campos**, debemos saber que las **interacciones con modelos y registros** se realizan a través de *recordsets*, que son una **colección ordenada** de registros del **mismo modelo**. La iteración en un recordset producirá nuevos conjuntos de un solo “**registro activo**” (**singletons o solteros**) del que se podrá **leer y escribir directamente**:

Recorriendo los registros

```
def do_operation(self):
    print(self) # => a.model(1, 2, 3, 4, 5)
    for record in self:
        print(record) # => a.model(1), then a.model(2), ...
```

También, se puede acceder a los valores de los campos **como elementos diccionario**. Establecer el **valor de un campo** desencadena una **actualización de la base de datos**:

Accediendo al campo por medio de elementos diccionario







```
>>> record.name = "Bob"
>>> field = "name"
>>> record[field]
```

Bob

Para ver los campos que tiene un modelo en Odoo, bastará con ir al modelo concreto y pulsar sobre él.

## Campos de un modelo

Nombre de campo	Etiqueta de campo	Tipo de campo	Requerido	Sólo lectura	Indexado	Tipo
activity_date_deadline	Siguiente plazo de actividad	fecha	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	Campo base 
activity_exception_decoration	Decoración de Actividad de Excepción	Selección	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	Campo base 
activity_exception_icon	Icono	Carácter	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	Campo base 
activity_ids	Actividades	one2many	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	Campo base 

## Archivos de datos

Cuando creamos un módulo de Odoo, se pueden **definir datos** que se guardarán en la base de datos. Estos datos pueden ser **necesarios tanto para:**

- el **funcionamiento** del módulo,
- como de **demostración**,
- o, incluso, parte de la **vista**.

Todos los **archivos de datos** están en **formato XML** con elementos `<record>`, siendo cada elemento `<record>` un **registro** de base de datos. Por ejemplo, en el módulo 'Empleados', habrá una **carpeta** '**data**' con un XML con todos los datos de todos los empleados. Los XML tienen una estructura básica como la siguiente:

Estructura básica de un archivo de datos

```
<odoo>

  <record model="{model name}" id="{record identifier}">
    <field name="{a field name}">{a value}</field>
  </record>

</odoo>
```

Donde:

- `model` es el nombre del **modelo** Odoo para el registro. Es **obligatorio** indicar uno.
- `id` es un **identificador externo**, permite hacer referencia al registro. **Se recomienda encarecidamente** informarlo.
- Los `fields` tienen un **atributo** '`name`', y es el nombre del campo en el modelo (ejemplo: descripción). Su **cuerpo será el valor** de dicho campo. Es **obligatorio** informar la **etiqueta** '`name`'.

Los **archivos de datos** deben **declararse en el** archivo **manifest** del módulo.

Pueden declararse en:

- la lista de '**datos**' (**siempre** cargada),
- o en la lista de '**demostración**' (solo cargados en modo de demostración).

Incluyendo los archivos de datos en el `__manifest__.py`

```
'data': [  
    'archivo_de_datos.xml',  
    'otro_archivo_de_datos.xml',  
    'otro_archivo_mas_de_datos.csv'  
]
```

Estos archivos XML son **flexibles y autodescriptivos**, y **muy detallados** cuando se crean varios registros simples del mismo modelo en masa.

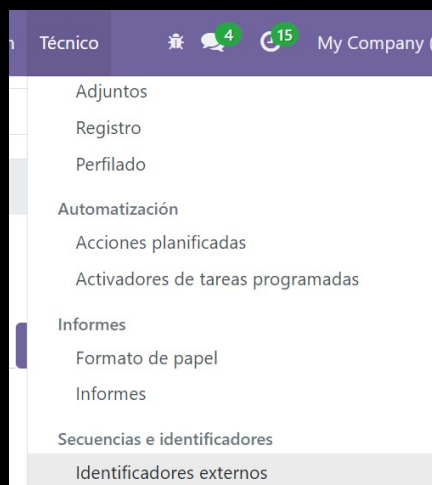
Para este caso, los archivos de datos **también pueden usar el formato CSV**, sabiendo que:

- el nombre del archivo será **model\_name.csv**.
- la **primera fila** enumera los **campos** para escribir.
- cada fila crea **posteriormente** un nuevo **registro**.

Todos los **'record'** de la base de datos tienen un **identificador único** en la tabla, el **id**. Es un **contador automático** asignado por la base de datos. Sin embargo, si queremos hacer referencia a él en ficheros de datos u otros lugares, no siempre tenemos por qué conocerlo. La solución de Odoo son los **IDs externos**, que es una **tabla que relaciona cada ID de cada tabla** con un **nombre**. Se trata del modelo `ir.model.data`.

Para encontrarlos, hay que ir a:

**Settings > Technical > Sequences & identifiers > External Identifiers**



Ahí encontramos la **columna Complete ID**.

Nuevo Identificadores externos ⚙		1-80 / 10000+ < > 🔍	
<input type="checkbox"/>	Id. completo	Nombre mostrado	Nombre del modelo ID de registro
<input type="checkbox"/>	account.1_capital	301000 Capital	account.account 22
<input type="checkbox"/>	account.1_cash_diff_expense	642000 Pérdida por diferenc...	account.account 36
<input type="checkbox"/>	account.1_cash_diff_income	442000 Ganancia por difere...	account.account 26

## Vistas, relaciones y herencias

### Vistas básicas

Las vistas son la forma en la que se representan los registros de los modelos para ser **mostradas a los usuarios finales**. Se especifican en **XML**, lo que significa que son **flexibles**, permiten un alto nivel de **personalización** y pueden **editarse independientemente** de los modelos que representan.

Cada vista se materializa en una **visualización distinta**:

- vistas de árbol,
- formularios,
- **kaban** -que son una mezcla de xml, html y plantillas Qweb-,
- de búsqueda,
- calendario
- o gráficos).

En caso de que **no declaremos** las vistas, se pueden **referenciar por su tipo** y Odoo generará una **vista de lista o formulario estándar** para poder ver los registros de cada modelo.

Las vistas **se declaran** como un **registro del modelo ir.ui.view**, por ejemplo:

Estructura básica de una vista en Odoo

```
<record model="ir.ui.view" id="view_id">
  <field name="name">view.name</field>
  <field name="model">object_name</field>
  <field name="priority" eval="16"/>
  <field name="arch" type="xml">
    <!-- view content: <form>, <tree>, <graph>, ... -->
  </field>
</record>
```

El **tipo de vista** está **implícito** en el **campo 'arch'**, concretamente en la raíz.

## Relaciones entre modelos

Un registro de un modelo puede estar **relacionado con un registro de otro modelo**. Las relaciones en Odoo se llaman:

- **many2one**: si es de 'muchos a 1'.
- **one2many**: las '1 a muchos'.
- **many2many**: 'muchos a muchos'.

Como sabemos, las relaciones '**muchos a muchos**' en una base de datos relacional implican una tercera tabla intermedia, pero en Odoo no tenemos que preocuparnos de esto, ya que el **ORM simplifica las relaciones**. El mapeado de los objetos **detectará la relación** y **creará las tablas, claves y restricciones** de integridad necesarias.

## Herencia

En el siguiente audio, hablaremos sobre las herencias de modelos en Odoo y cómo puede afectar a cada componente del MVC (modelo-vista-controlador):

### Las herencias en Odoo

El framework de Odoo ofrece el mecanismo de la herencia para que los programadores puedan **adaptar módulos existentes** y garantizar a la vez que las **actualizaciones** de los módulos **no rompan los originales**.

La herencia se puede **aplicar en los tres componentes del MVC**:

- En el **modelo**: posibilita **ampliar las clases** existentes o **diseñar nuevas** clases a partir de las existentes.
- En la **vista**: posibilita **modificar el comportamiento** de vistas existentes o **diseñar nuevas** vistas.
- En el **controlador**: posibilita **sobrescribir los métodos** existentes o diseñar **otros nuevos**.

Odoo proporciona **tres mecanismos de herencia**:

1. la herencia de **clase**,
2. la herencia para **prototipo**,
3. y la herencia **por delegación**.

## Informes y controladores

### Los informes o reports

Los informes son los archivos que imprimiremos, almacenaremos o enviaremos por e-mail. El motor de informes utiliza una combinación de [Twitter Bootstrap](#), [QWeb](#) y [Wkhtmltopdf](#).

Consta de dos partes:



- Un `ir.actions.report` para el que se suministra un **elemento de acceso <report>** y establece varios **parámetros** básicos para el informe:
  - como el **tipo predeterminado**,
  - si el informe debe **almacenarse en base de datos** tras la creación,
  - etc.
- Una vista `Qweb` para el contenido.




Ya que los informes son **páginas web estándar**, se puede acceder a ellos a través de una **URL**. Además, los **parámetros de salida** se pueden **modificar** a través de esta URL, por ejemplo, la **versión html** del informe de factura es accesible por medio de la siguiente URL:


[http://localhost:8069/report/html/account.report\\_invoice/1](http://localhost:8069/report/html/account.report_invoice/1)

y la **versión en PDF** por medio de la siguiente:

[http://localhost:8069/report/pdf/account.report\\_invoice/1](http://localhost:8069/report/pdf/account.report_invoice/1)

1 / 1 | - 97% + |  



My Company (San Francisco)

Gran vía 14

28200 Madrid (Madrid)

España

Azure Interior

4557 De Silva St

Fremont CA 94538

Estados Unidos

Identificación fiscal: US12345677

Factura INV/2024/00001

Fecha de factura:

01/01/2024

Fecha de vencimiento:

29/02/2024

Fecha de envío:

01/01/2024

Descripción	Cantidad	Precio Unitario	Impuestos	Cantidad
[FURN_6741] Gran mesa de reuniones Mesa de la sala de conferencias	5,00	4.000,00	15 %	\$ 20.000,00
[FURN_8220] Escritorio para cuatro personas Estación de trabajo de oficina moderna para cuatro personas	5,00	2.350,00	15 %	\$ 11.750,00
Condiciones de pago: fin del siguiente mes	Importe neto			\$ 31.750,00
Comunicaciones de pago INV/2024/00001	Impuesto del 15 %			\$ 4.762,50
	Total			\$ 36.512,50



## Los controladores

Todos los frameworks suelen tener controllers, y Odoo tiene los suyos. Los controladores de un módulo de Odoo son **clases** capaces de **capturar las solicitudes http** enviadas por cualquier navegador y que se usan cuando un **usuario quiere procesar algunos datos** en un sitio web. En Odoo, los controladores se utilizan para **configurar los módulos frontend**. Estos módulos frontend están **integrados con** módulos **backend**.

Por ejemplo, un usuario no puede usar la funcionalidad de “Models” en Odoo para representar los detalles del pedido de ventas en el sitio web. Sin embargo, **al usar el controlador**, los usuarios pueden **obtener** fácilmente los **detalles** del pedido de ventas **del backend**.

Los módulos como “**blog** del sitio web”, “**venta** del sitio web” y “**foro** del sitio web” están utilizando **controladores para ampliar sus funcionalidades**.

Los controladores se crean heredando del **Controller**.

Las **rutas** se definen a través de **métodos decorados con route()**:

Creando un controlador

```
from odoo import http

class MyController(odoo.http.Controller):
    @route('/some_url', auth='public')
    def handler(self):
        return self.stuff()

    def stuff(self):
        # Aquí deberías colocar la lógica de tu controlador
        # Puede ser cualquier cosa que desees devolver como respuesta
        return "Hello from MyController!"
```

Se utiliza el decorador `@route` para definir la ruta (`/some_url`) y la **autenticación** (`auth='public'`) para el método `handler`.

Con los controladores se puede realizar solicitudes http, solicitudes json, enrutamientos, renderizados, etc.

## Campos calculados

Es posible que, a la hora de mostrar los campos, **en lugar de traernos el valor de la base de datos**, el campo sea un campo calculado (**haciendo uso de otros campos**).

El valor del campo no se recupera de la BBDD, sino que se calcula sobre la marcha **invocando a un método del modelo**.

Para implementar este tipo de campos, bastará con **crear un campo** e informar su **atributo 'compute'** con el **nombre de un método**:

Ese método será el método de **cálculo**, que deberá establecer el **valor del campo** para calcular en cada registro en el atributo **'self'**.

Creando un **campo calculado** llamado **name**

```
import random
from odoo import models, fields, api

class ComputedModel(models.Model):
    _name = 'test.computed'

    name = fields.Char(compute = '_compute_name')

    def _compute_name(self):
        for record in self:
            record.name = str(random.randint(1, 1e6))
```

Explicación de este código:

**\_name**: Especifica el nombre del modelo en Odoo.

**name**: Este es el campo calculado. Se define como un campo de tipo Char y utiliza el decorador `@api.depends()` para indicar que el método `_compute_name` es el encargado de calcular su valor.

**\_compute\_name**: Este método calcula el valor del campo name. En este caso, simplemente genera un número aleatorio entre 1 y 1 millón y lo convierte a cadena para asignarlo al campo name del registro actual.

Recuerda que cuando se accede al campo name, automáticamente se ejecuta el método `_compute_name` para calcular su valor. Si bien este ejemplo utiliza un valor aleatorio, en situaciones más prácticas, este método podría depender de otros campos en el mismo modelo o incluso en modelos relacionados, permitiendo cálculos más complejos y específicos según las necesidades del negocio.

## Configurando PyCharm para trabajar con Odoo

Para los dos siguientes temas (14 y 15), vamos a necesitar tener configurado el entorno de desarrollo PyCharm (con el que ya trabajamos en temas anteriores) para trabajar con Odoo.

Para ello, **desde PyCharm**, abriremos la **carpeta** donde está **instalado Odoo** con:

### File/Open

pero, ojo, no la carpeta que contiene el código fuente de Odoo (odoo.bin, odoo.conf, etc.),

sino **C:\Program Files (x86)\Odoo 13.0\server**.

A continuación, crearemos un launcher para lanzar Odoo cuando necesitamos. Para ello, en la parte superior derecha, pulsaremos sobre 'Add configuration'.


(En la versión 2023.3.3 no se ve esa pestaña, hay que buscar por: run/debug, y pulsar en la lista de resultados en "Edit Configuration ... run", y se abre la ventana, y pulsamos en el signo "+" de la izquierda superior. Ayuda: <https://www.youtube.com/watch?v=Uz8eguVc2LQ>).

Se nos abrirá una ventana en la que tendremos que informar:


- El nombre
- La ruta del script que queremos lanzar, que es el odoo.bin que tenemos en el rooth.
- El intérprete de Python; en nuestro caso, seleccionaremos la 3.7.
- Los parámetros, que informaremos con "--conf="C:\Program Files (x86)\Odoo 13.0\server\odoo.conf"" para que seleccione correctamente el fichero 'odoo.conf'. (Sustituir la ruta por la nuestra, al archivo odoo.conf, evidentemente).

## Configurando el launcher



Name:



☐ Store as project file 

Run Modify options ▾ Alt+M

 Python 3.12 (server) C:\Program Files\Odoo 17.0.20240107\server\venv\Scripts\python.exe ▾

script ▾

C:\Program Files\Odoo 17.0.20240107\server\odoo-bin  

--conf="C:\Program Files\Odoo 17.0.20240107\server\odoo.conf"  

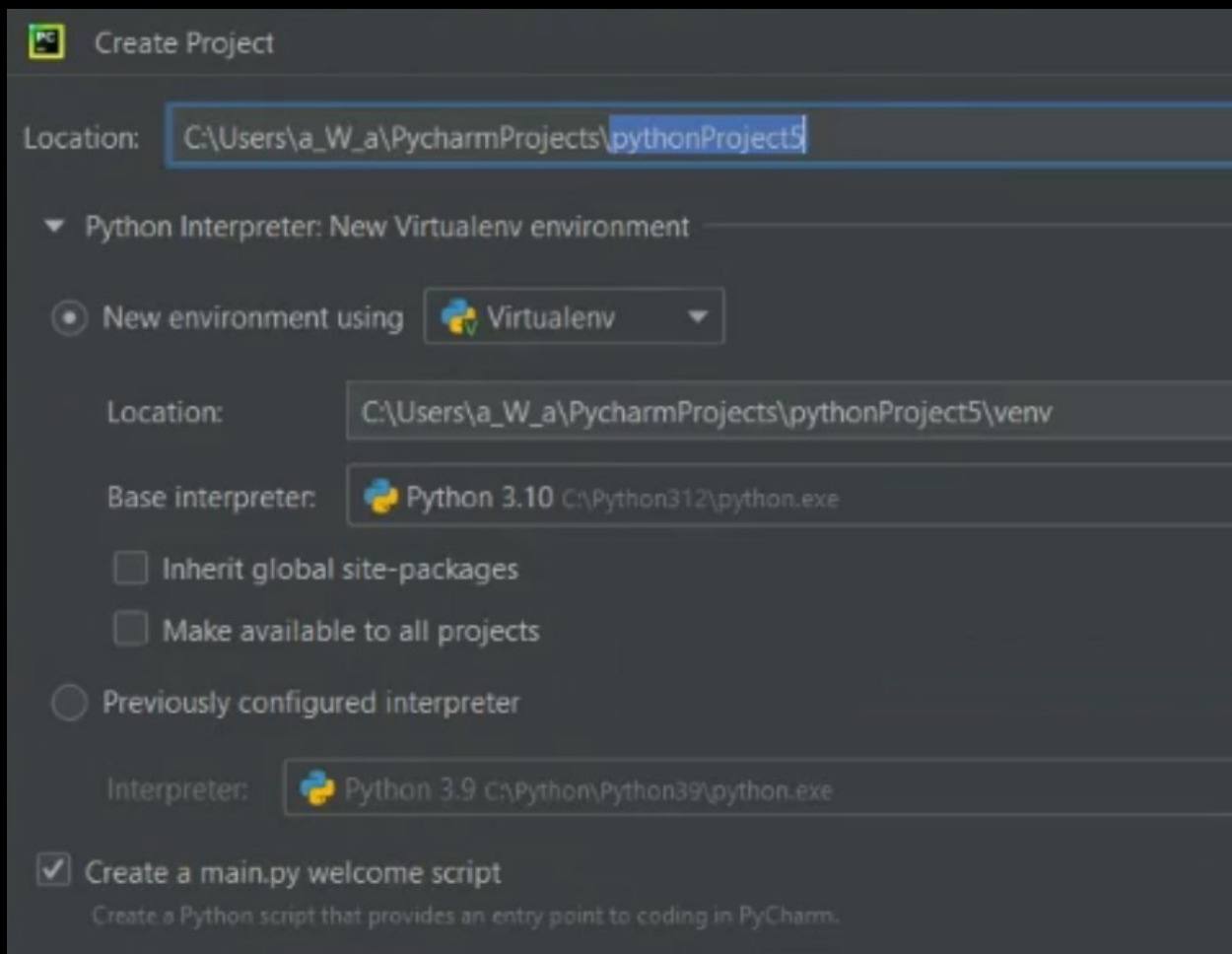
Press Alt for field hints

Al pulsar sobre el icono play, comprobaremos que la conexión funciona correctamente según el log. No debemos alarmarnos de que aparezcan las letras en rojo.

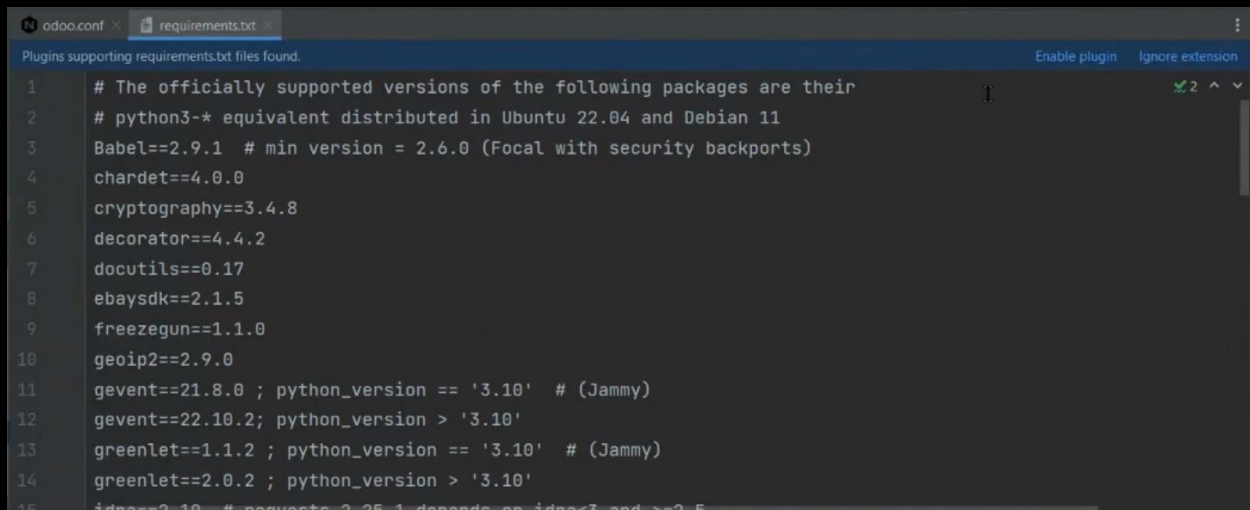
## EXPLICACIÓN DEL PROFESOR: CONFIGURACIÓN DE PYCHARM CON ODOO

Vamos a abrir el PyCharm y les digo, si yo quiero enganchar el PyCharm con mi entorno de desarrollo, con el servidor, lo que tengo que hacer es:

1. Nuevo proyecto.
2. Al **nuevo proyecto**, decirle que vaya a la ruta donde tengo instalado Odoo, que en este caso es aquí, hasta **server**:  
C:\Program Files\Odoo 17.0.20231128\server  
Siguiendo las indicaciones, navego y llego hasta server.
3. Tengan cuidado con el **intérprete** que se pone, si no supera la versión 3.10 no le va a arrancar, entonces tienen que tener en cuenta eso.

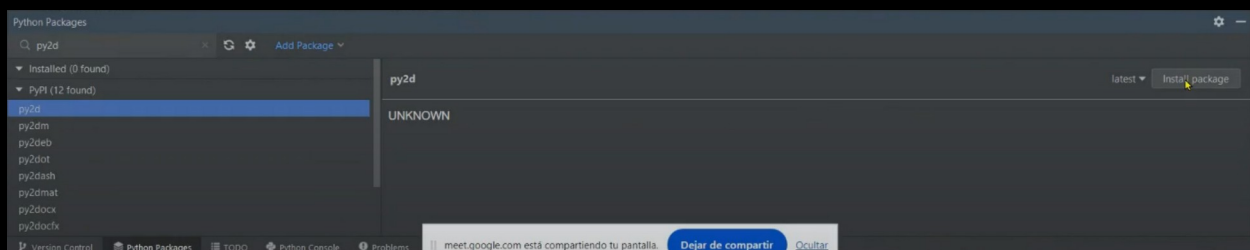


Una vez que lo hacen, puede que le dé algún problema de que necesitan paquetes instalados, entonces hay un fichero que se llama `requirements.txt`, donde aquí tiene todos los requisitos que se necesita para poder arrancar. Le aparece: “instalar requisitos o requerimientos”, y empiezan a descargarse todos estos paquetes. He tenido algún problema con este de aquí, que he tenido que cambiarle la versión (`#Werkzeug==2.0.2` → `Werkzeug==2.2.2`).

A screenshot of a code editor window with two tabs: 'odoo.conf' and 'requirements.txt'. The 'requirements.txt' tab is active, showing a list of Python dependencies. The text is as follows:

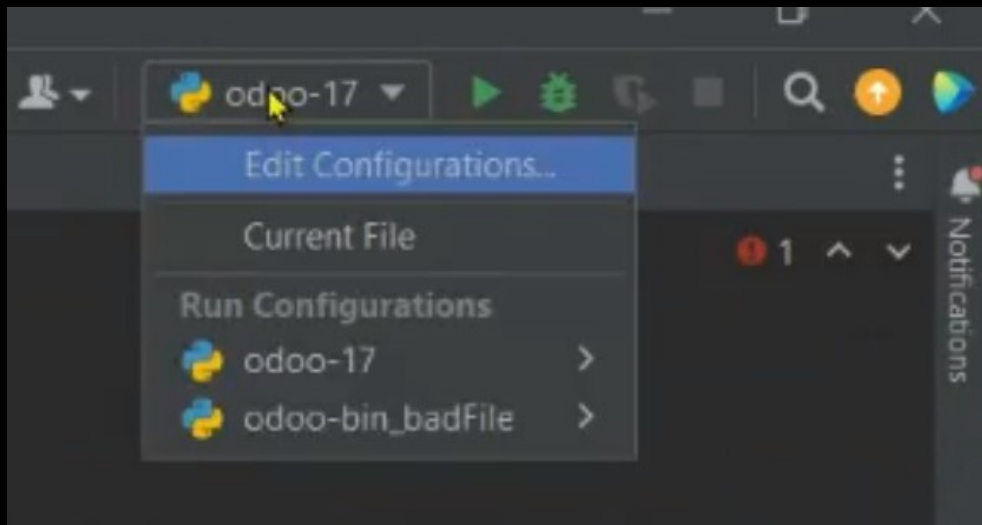
```
1 # The officially supported versions of the following packages are their
2 # python3-* equivalent distributed in Ubuntu 22.04 and Debian 11
3 Babel==2.9.1 # min version = 2.6.0 (Focal with security backports)
4 chardet==4.0.0
5 cryptography==3.4.8
6 decorator==4.4.2
7 docutils==0.17
8 ebaysdk==2.1.5
9 freezegun==1.1.0
10 geoip2==2.9.0
11 gevent==21.8.0 ; python_version == '3.10' # (Jammy)
12 gevent==22.10.2; python_version > '3.10'
13 greenlet==1.1.2 ; python_version == '3.10' # (Jammy)
14 greenlet==2.0.2 ; python_version > '3.10'
15 idna==2.10 # requests 2.25.1 depends on idna<3 and >=2.5
```

Y si ustedes necesitan **hacerlo a mano**, porque hay algún **paquete que no está aquí**, lo pueden añadir y volver a lanzar, sabiendo qué paquete es y con qué versión. Si no, se vienen aquí a Python Packages, y pueden buscar por ejemplo: `py2d`, lo que vayan queriendo y le dicen si está instalado o no está instalado, y si no lo tienen instalado, le dicen instalar paquete, y se va instalando para resolver todas las dependencias:



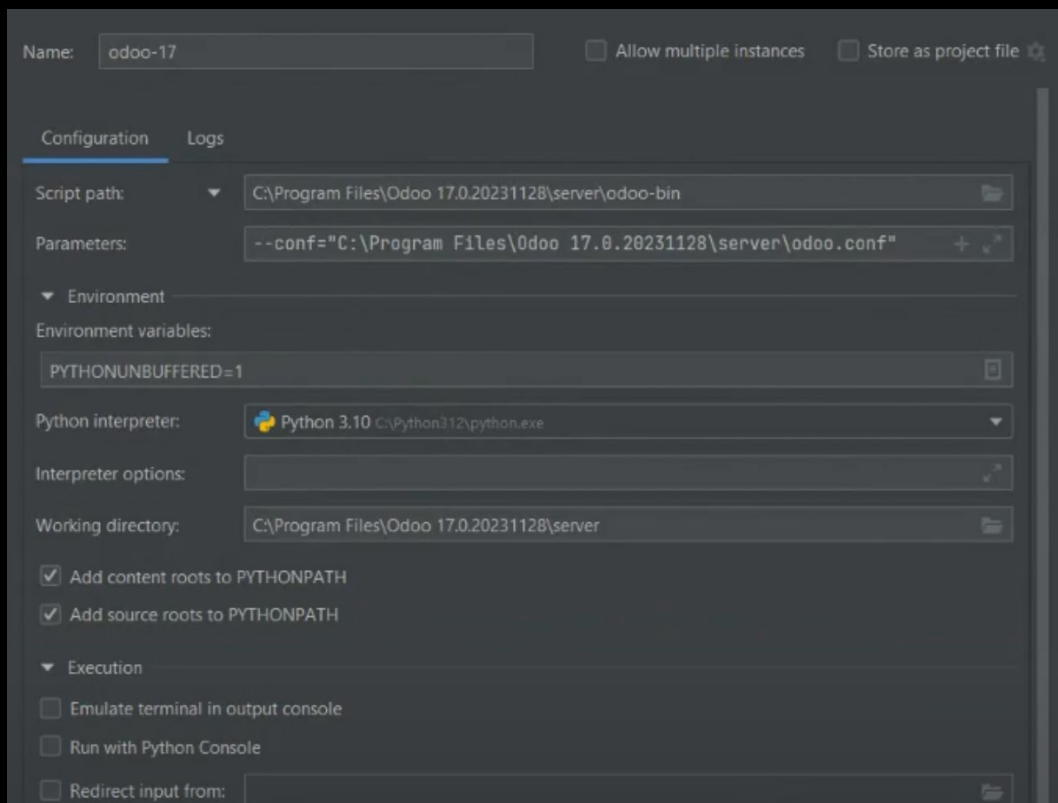
Una vez que está esto así, le dan a play.

Hay que configurar el lanzador, aquí:



Todos los datos que vienen puestos en el tema se aproximan y le sale:

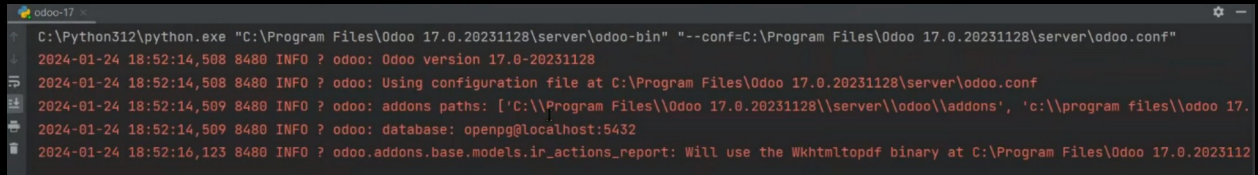
- el path, a donde lo tengan, de odoo-bin,
- le tienen que poner este parámetro, indicándole dónde está odoo.conf,
- tienen que poner este Environment, esta variable: PYTHONUNBUFFERED=1.
- El intérprete, insisto que la versión tiene que ser la apropiada.
- Una vez configurado, le dan a arrancar.



Tienen que ponerlo como archivo Python.

Entonces, cuando quiera arrancar, empezará a informar aquí de si está el servidor.

Qué está cargando... Qué está haciendo...



```
C:\Python312\python.exe "C:\Program Files\Odoo 17.0.20231128\server\odoo-bin" "--conf=C:\Program Files\Odoo 17.0.20231128\server\odoo.conf"
2024-01-24 18:52:14,508 8480 INFO ? odoo: Odoo version 17.0-20231128
2024-01-24 18:52:14,508 8480 INFO ? odoo: Using configuration file at C:\Program Files\Odoo 17.0.20231128\server\odoo.conf
2024-01-24 18:52:14,509 8480 INFO ? odoo: addons paths: ['C:\Program Files\Odoo 17.0.20231128\server\odoo\addons', 'c:\program files\odoo 17.0.20231128\server\odoo\addons']
2024-01-24 18:52:14,509 8480 INFO ? odoo: database: openpg@localhost:5432
2024-01-24 18:52:16,123 8480 INFO ? odoo.addons.base.models.ir_actions_report: Will use the Wkhtmltopdf binary at C:\Program Files\Odoo 17.0.20231128\server\odoo\addons\wkhtmltopdf.exe
```

Y cuando hagamos peticiones sobre el localhost, la petición que vamos a hacer al servidor, responde aquí con los mensajes.



## Navegando por Odoo con PyCharm

Veremos cómo movernos por Odoo con la herramienta, con el software PyCharm. Después de haber configurado el proyecto con la carpeta de Odoo y haber configurado el launcher aquí, ya estamos en disposición de realizar modificaciones en PyCharm que se verán reflejadas en Odoo. Por ejemplo, aquí vemos el `odoo.conf`, que viene con la configuración de Odoo que le hemos indicado, el `odoo.bin` que será el ejecutable vinculado al launcher que será el que ejecutará nuestro Odoo. Si le damos al play, aquí veremos en un log que se conecta correctamente a nuestro Odoo. Para aprender un poco más a movernos por aquí, podemos ver en la carpeta de Odoo, aquí nos encontramos los módulos, en cuyo interior, por ejemplo, en el módulo `product` nos encontramos con la carpeta del modelo:

En primer lugar vemos que tiene tanto el manifest informado de su descripción, su nombre, su categoría de descripción, de dónde se traen los datos y demás, como su `__init__.py`, que en este caso tiene tres imports.

También nos encontramos aquí con los **modelos**, en cuyo caso, por ejemplo, en el modelo `Product Template` del módulo `Product`, podemos encontrarnos con la declaración de los campos, por ejemplo. Aquí es muy útil esta herramienta Structure (a la derecha) que te enseña la clase con todos sus métodos, los campos y demás.

Aquí podemos ver, por ejemplo, la **declaración de los campos en el Product Template** y además podemos buscar relación entre modelos, one to many, por ejemplo. Aquí vemos que este campo, `Packet It`, está relacionado con otro modelo, con un one to many.

También vemos **campos calculados** con la estructura `Compute`. Vemos que `Packet It` o `Packet`, este campo `Valid Product` al tener un **compute** es un campo calculado. Después de ver los modelos y los campos y las relaciones entre los modelos, campos calculados y demás, también podemos ver los datos. Aquí, como podemos observar, hay dos archivos `.xml` con los **datos record**.

Después también tenemos aquí las **vistas**. Todo esto dentro del módulo de `Product`. Aquí tenemos las vistas, las diferentes vistas de este módulo, los report, que también hemos hablado de ellos en este tema. En este caso no tiene carpeta de controles, pero, por ejemplo, el módulo `Project` sí tiene aquí su **carpeta de controles**. Con sus enrutamientos y demás.

Y con esto hemos hecho un repaso breve sobre lo que hemos tratado en este tema y visualizándolo desde PyCharm de Odoo.

## Reflexión

Odoo tiene una serie de objetos interrelacionados entre sí, ajenos al usuario final, que hacen que disponga de esa interfaz gráfica tan sólida y amigable.

**Cada módulo de Odoo tiene en su interior uno o varios modelos**, que son archivos Python que relacionan el programa con la base de datos. Estos se ayudan de los métodos que ofrece la capa ORM para evitar tener que realizar consultas SQL. Además, deben tener vistas, que son archivos XML que representarán los registros de los modelos para ser mostrados a los usuarios finales. Esos registros están almacenados en los archivos de datos, que son documentos XML con elementos <record>, siendo cada uno de estos elementos un registro de base de datos. Además, existe la posibilidad de relacionar modelos, de implementar informes, campos calculados o crear controladores.

[https://www.odoo.com/es\\_ES/https://castilloinformatica.es/wiki/index.php?title=Odoo](https://www.odoo.com/es_ES/https://castilloinformatica.es/wiki/index.php?title=Odoo)

<http://vauxoo.github.io/>