Creación de módulos en Odoo.

Comprobamos que odoo está corriendo como servicio e iniciamos una shell con el usuario operador_odoo que creamos con la instalación de odoo.

sxe@sxe-VirtualBox:~$ systemctl status odoo16

sxe@sxe-VirtualBox:~$ su - operador_odoo //abc123..

Tenemos una carpeta que se llama addons donde están cada una de las carpetas de los módulos de odoo.

```
operador_odoo@sxe-VirtualBox:~$ ls -l
total 116
drwxr-xr-x 474 operador_odoo operador_odoo 20480 oct  7 16:11 addons
-rw-r--r--   1 operador_odoo operador_odoo   803 oct  7 16:11 CONTRIBUTING.md
-rw-r--r--   1 operador_odoo operador_odoo   433 oct  7 16:11 COPYRIGHT
drwxr-xr-x   3 operador_odoo operador_odoo  4096 oct  7 16:11 debian
drwxr-xr-x   3 operador_odoo operador_odoo  4096 oct  7 16:11 doc
-rw-r--r--   1 operador_odoo operador_odoo 43529 oct  7 16:11 LICENSE
-rw-r--r--   1 operador_odoo operador_odoo   124 oct  7 16:11 MANIFEST.in
drwxr-xr-x  12 operador_odoo operador_odoo  4096 oct  7 23:05 odoo
-rwxr-xr-x   1 operador_odoo operador_odoo   180 oct  7 16:11 odoo-bin
-rw-r--r--   1 operador_odoo operador_odoo  2114 oct  7 16:11 README.md
-rw-r--r--   1 operador_odoo operador_odoo  3021 oct  7 16:11 requirements.txt
-rw-r--r--   1 operador_odoo operador_odoo  1734 oct  7 16:11 SECURITY.md
drwxr-xr-x   4 operador_odoo operador_odoo  4096 oct  7 16:11 setup
-rw-r--r--   1 operador_odoo operador_odoo   339 oct  7 16:11 setup.cfg
-rw-r--r--   1 operador_odoo operador_odoo  1759 oct  7 16:11 setup.py
```

Por poner un ejemplo, la carpeta que vemos a continuación tiene toda la estructura para el funcionamiento del módulo de “punto de ventas” o “point_of_sale” en inglés.

1. Preparando carpeta para guardar módulos.

Los módulos que vienen con Odoo o que vamos descargando se alojan en la carpeta /opt/odoo/addons pero los que creemos o adaptemos nosotros los vamos a dejar en una carpeta distinta. Además de una mejor organización esto ayuda a que tengamos claro y podamos distinguir los módulos “oficiales” de los propios. En caso de una reinstalación de Odoo nuestros módulos no serán sobreescritos. También resultará interesante si queremos llevarlos a otra instancia de Odoo en otro equipo.

```
mkdir /opt/odoo/modulos_extra
```

Este comando crea una carpeta llamada modulos_extra. Usaremos esta carpeta para dejar nuestros módulos.

```
nano /opt/odoo/.odoorc
```

Este comando abre para editar el archivo de configuración .odoorc. En este archivo debemos buscar la línea addons_path y dejarla de la siguiente manera:

```
addons_path = /opt/odoo/addons,/opt/odoo/modulos_extra
```

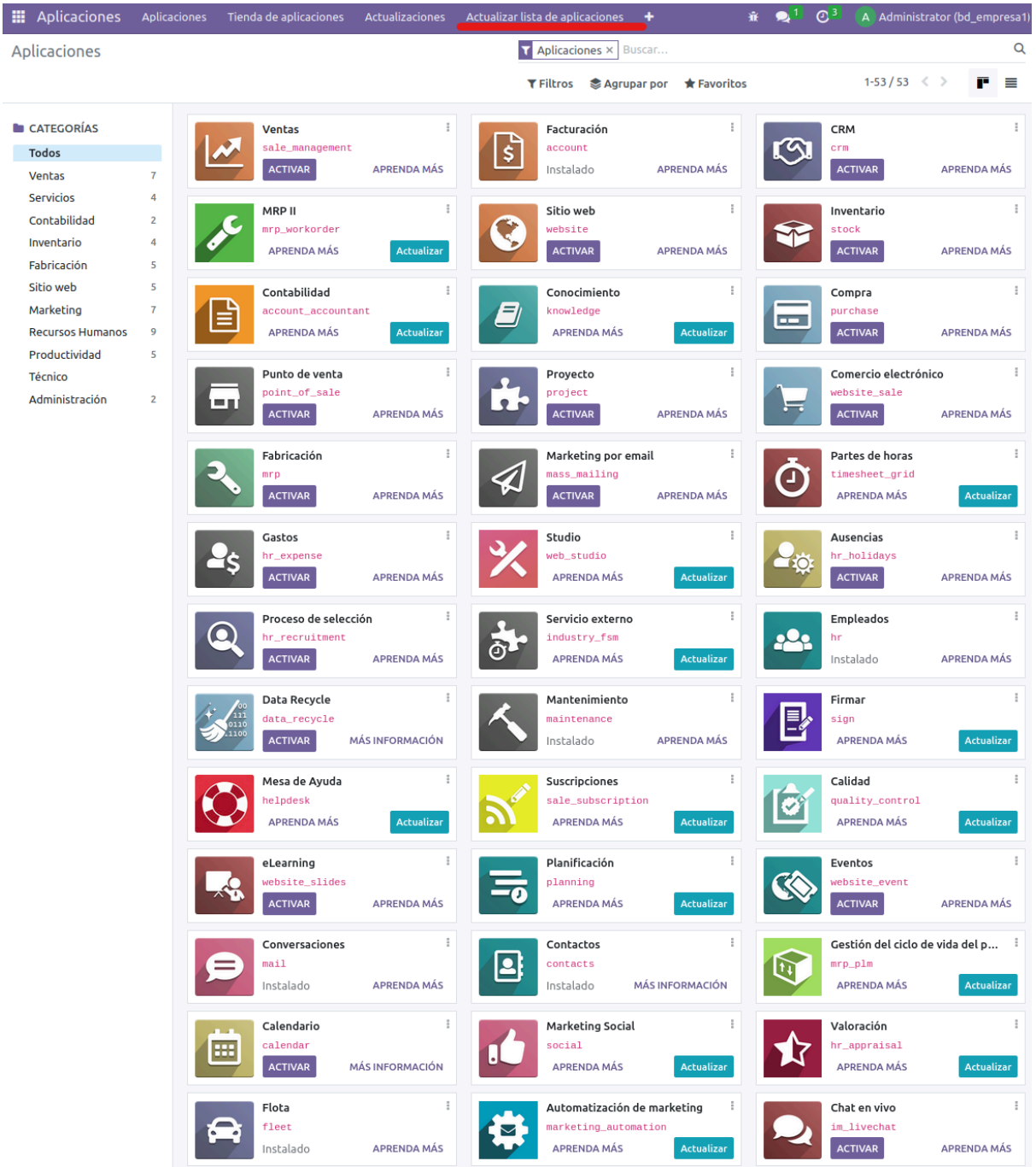
Ojo con separar rutas con coma pero no introducir espacios. Guardamos con Ctrl + X. Con este cambio en el archivo de configuración hemos informado a Odoo que puede buscar módulos en la carpeta por defecto addons y también en módulos_extra.

```
exit
```

```
sudo systemctl restart odoo16
```

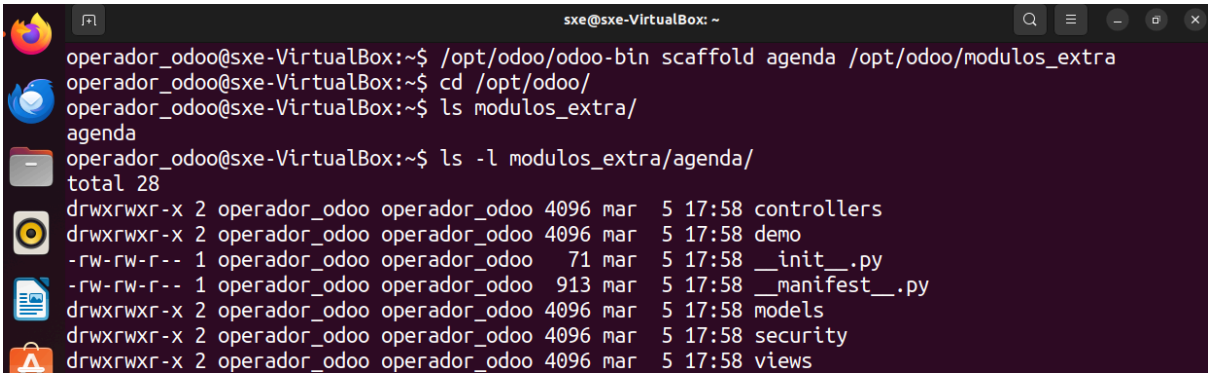
Con estos dos comandos salimos del usuario operador_odoo y **reinciamos** Odoo para que acepte los cambios. En este punto tenemos que ir a Odoo y clicar en modo desarrollador (Ajustes/Activar modo de desarrollador). Con esto garantizamos que se busquen nuevos módulos en la carpeta de addons y en la carpeta de modulos_extra.

2. Crear estructura del módulo

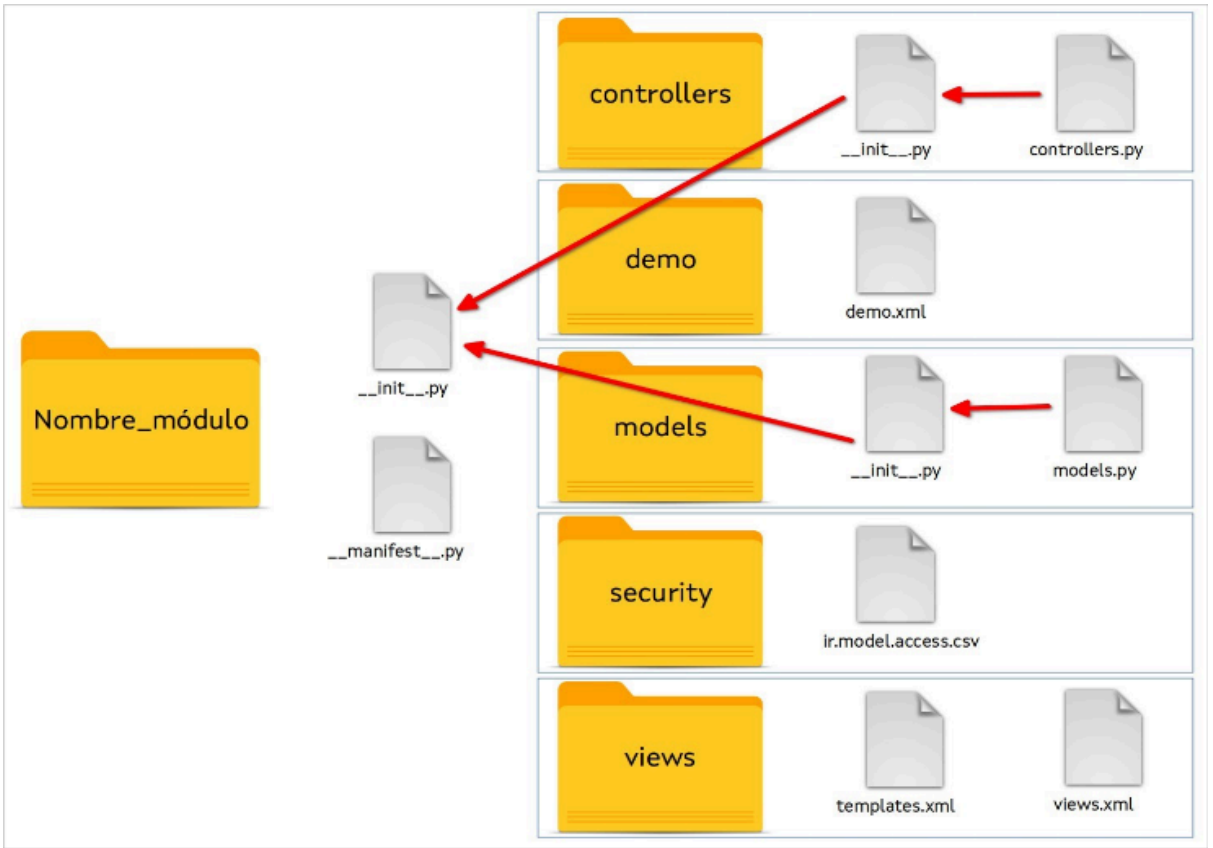


```
/opt/odoo/odoo-bin scaffold agenda /opt/odoo/modulos_extra
```

Con este comando ejecutamos una instancia de odoo-bin con el modificador **scaffold** que sirve para construir la estructura de un módulo en Odoo. Para ellos también hay que pasar por argumentos el nombre del módulo, agenda, y la ruta de la carpeta donde se guardará.



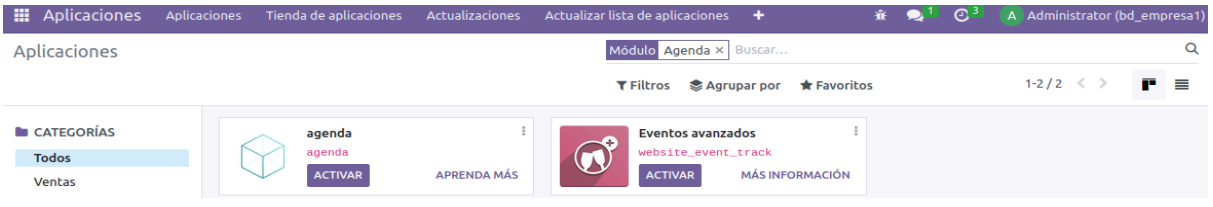
A continuación se adjunta un diagrama de la estructura que se ha generado de este nuevo módulo.

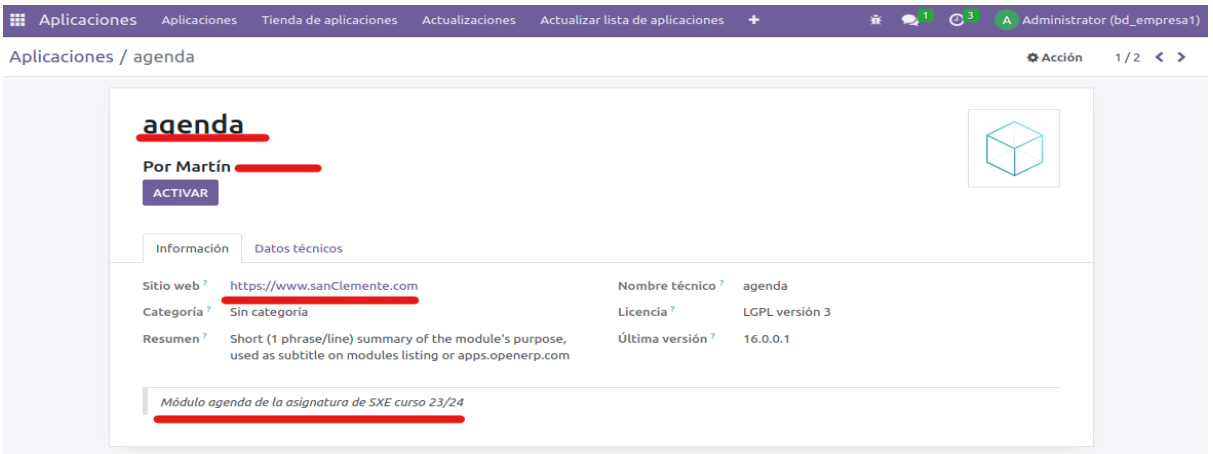


Cada módulo de Odoo se identifica mediante el archivo **__manifest__.py** que está en la raiz de las carpetas del módulo. Técnicamente es un diccionario de python, es decir un conjunto de pares de clave-valor separados por comas. En este archivo puedes encontrar metadatos o información administrativa del módulo como la autoría, la licencia o el precio que mostrará en la tienda de aplicaciones de Odoo. Será necesario editar el módulo **__manifest__.py** para que este tenga el comportamiento que nosotros deseamos.

```
 GNU nano 6.2                    agenda/__manifest__.py *
# -*- coding: utf-8 -*-
{
    'name': "agenda",

    'summary': """
        Short (1 phrase/line) summary of the module's purpose, used as
        subtitle on modules listing or apps.openerp.com""",

    'description': """
        Módulo agenda de la asignatura de SXE 23/24
    """,

    'author': "Martín Gil",
    'website': "https://www.yourcompany.com",

    # Categories can be used to filter modules in modules listing
    # Check https://github.com/odoo/odoo/blob/16.0/odoo/addons/base/data/ir_m>
    # for the full list
    'category': 'Uncategorized',
    'version': '0.1',

    # any module necessary for this one to work correctly
    'depends': ['base'],

    # always loaded
    'data': [
        'security/ir.model.access.csv',
        'views/views.xml',
        'views/templates.xml',
    ],
    # only loaded in demonstration mode
    'demo': [
        'demo/demo.xml',
    ],
}
```

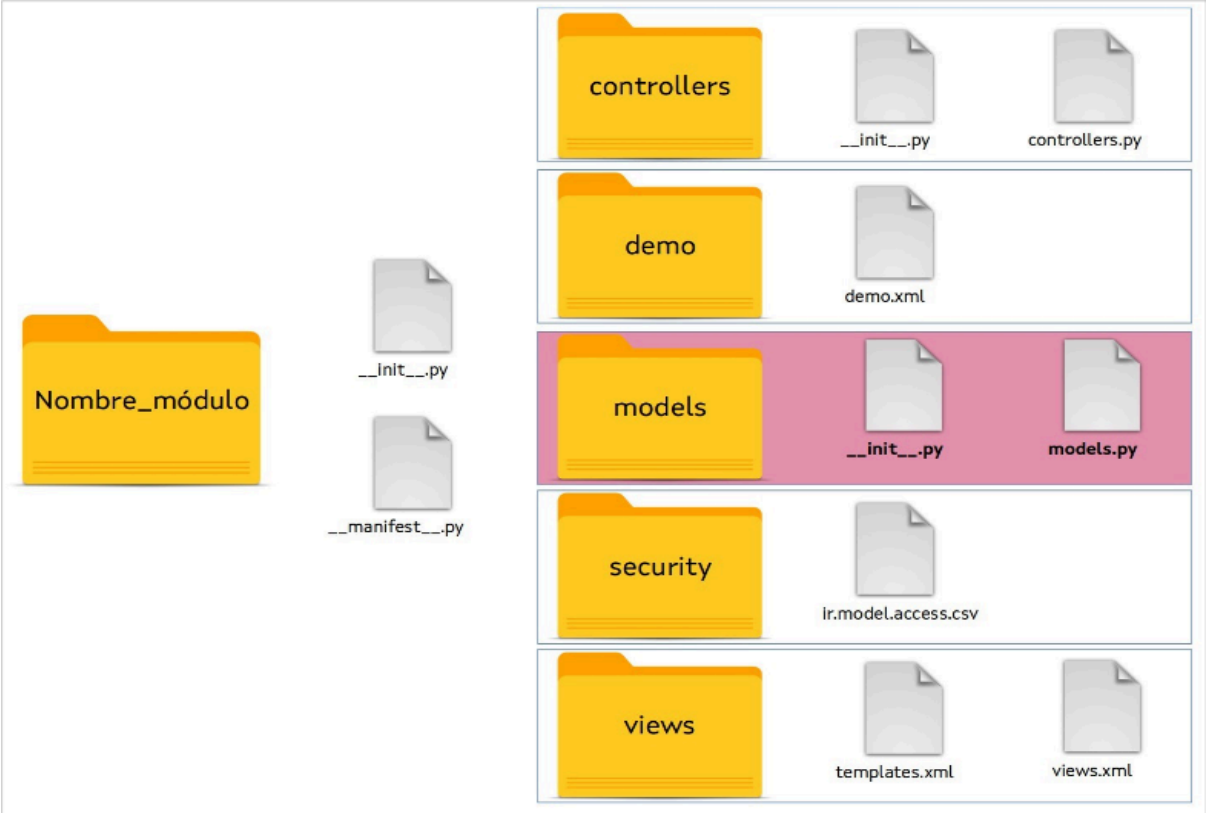
Llegados a este punto tenemos que **reiniciar** Odoo y podemos buscar nuestro módulo en Odoo. ***Es importante actualizar la lista de aplicaciones y eliminar el filtro de "Aplicación" en el buscador para que aparezca.*** ***NO VAMOS A ACTIVAR AGENDA DE COMENTO.***





3. Los modelos

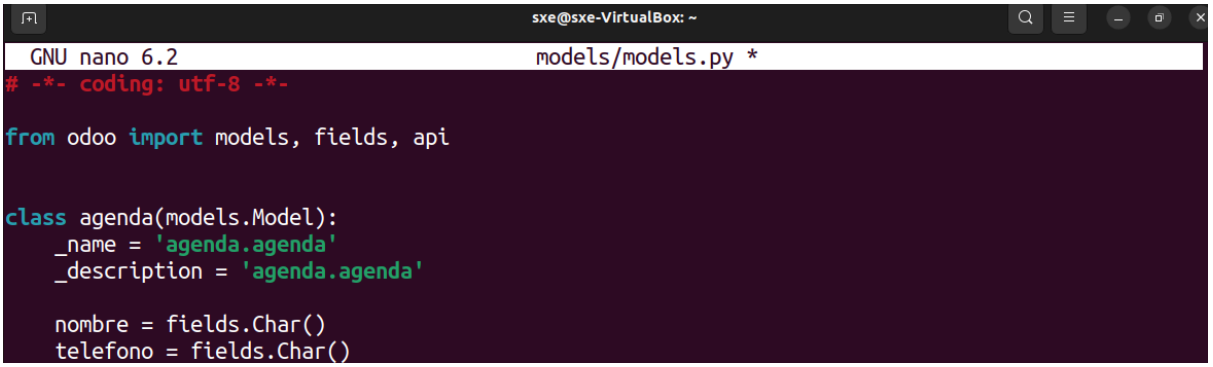
La lógica de negocio se programa en **modelos** que son clases python que extienden la **clase Model**. Los modelos creados pueden tener atributos, entre ellos **_name.** Este campo es obligatorio y representa el nombre del módulo, además se usa para crear una tabla con este mismo nombre en la base de datos.



Vamos ahora a editar el archivo models/models.py para crear la definición de la tabla que se va a crear en nuestra base de datos postgres. En este caso, vamos a crear una agenda que tenga dos atributos de tipo carácter, que van a ser el nombre y teléfono. Llegados a este punto podemos mencionar:

- La variable **_name** indica el nombre de la tabla que se va a crear en postgres. La tabla se va a llamar agenda.agenda. La primera agenda se refiere al nombre del módulo.
- La variable **_description** es simplemente un texto descriptivo que se almacena en BD, aquí podríamos poner otra información pero no es necesario.
- A continuación tendremos las columnas de la tabla que nosotros queremos tener para los datos.

```
 GNU nano 6.2                                models/models.py
# -*- coding: utf-8 -*-

# from odoo import models, fields, api


# class agenda(models.Model):
#     _name = 'agenda.agenda'
#     _description = 'agenda.agenda'

#     name = fields.Char()
#     value = fields.Integer()
#     value2 = fields.Float(compute="_value_pc", store=True)
#     description = fields.Text()
#
#     @api.depends('value')
#     def _value_pc(self):
#         for record in self:
#             record.value2 = float(record.value) / 100
```



Los tipos disponibles son:

-**Binary**. Usado para archivos (imágenes, pdfs, audio,…)

```
adjunto = fields.Binary()
```

-**Boolean**. Usado para valores de verdadero o falso.

```
tarea_finalizada = fields.Boolean()
```

-**Char**. Usado para cadena de caracteres.

```
nombre = fields.Char()
```

-**Date**. Usado para solo fechas o **DateTime** para fechas y horas.

```
fechaNacimiento = fields.Date()
cita_reunion = fields.DateTime()
```

-**Float**. Usado para valores decimales

```
altura = fields.Float()
```

-**Html**. Usado para código HTML

```
cabecera_web = fields.Html()
```

-**Integer**. Usado para valores enteros.

```
prioridad = fields.Integer()
```

-**One2many**. Usado para relaciones de uno a muchos. Se almacena en un array.

```
telefonos = fields.One2many()
```

-**Many2many**. Usado para relaciones de muchos a muchos.

```
tareas_empleados = fields.Many2many()
```

-**Many2one**. Usado para relaciones de muchos a uno. Muestra todos los registros pero almacena uno.

```
usuario = fields.Many2one()
```

-**Reference**. Usado para crear una relación con un modelo y una fila.

```
documento = fields.Reference()
```

-**Selection**. Usado para desplegar un conjunto de valores

```
provincia = fields.Selection()
```

-**Text**. Usado para cadena de caracteres larga.

```
observaciones = fields.Text()
```

A todos estos tipos de datos se le pueden introducir modificadores en los parámetros. Algunos de ellos son:
- **default**.Para dar valor por defecto o llamar a una función-

```
tarea_finalizada = fields.Boolean(default=False)
```

-**compute**. Llama a una función que calcula el dato.

```
total_factura = fields.Float(compute="_sumaDetalles")
```

-**related**. Un campo de sólo lectura llamado desde otra tabla.

```
nombre_en_factura = fields.Char(related='partner_id)
```

-**string**. El texto de la etiqueta que acompañará a este campo en la UI

```
nombre = fields.Char(string="Nombre completo")
```

-**required**. Indica obligatoriedad de cubrir el campo.

```
dni = fields.Char(required=True)
```

Resumiendo. Crear un modelo es crear una tabla en la base de datos con el nombre indicado en _name. Los campos que contendrá esa tabla es necesario definirlos usando los tipos anteriores con sus parámetros.

4. Las vistas

Las vistas se definen en archivos XML junto con las acciones y menús. Vamos a ir a la carpeta /views y a realizar modificaciones sobre views.xml. Vamos a modificar la vista para que quede de la siguiente manera:

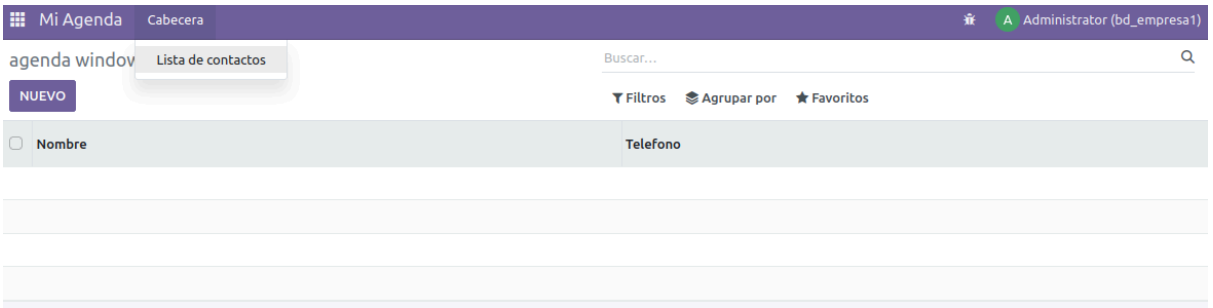

```
<odoo>
  <data>
    <!-- explicit list view definition -->

    <record model="ir.ui.view" id="agenda.list">
      <field name="name">agenda list</field>
      <field name="model">agenda.agenda</field>
      <field name="arch" type="xml">
        <tree>
          <field name="nombre"/>
          <field name="telefono"/>
        </tree>
      </field>
    </record>

    <!-- actions opening views on models -->

    <record model="ir.actions.act_window" id="agenda.action_window">
      <field name="name">agenda window</field>
      <field name="res_model">agenda.agenda</field>
      <field name="view_mode">tree,form</field>
    </record>

    <!-- server action to the one above -->
<!--
    <record model="ir.actions.server" id="agenda.action_server">
      <field name="name">agenda server</field>
      <field name="model_id" ref="model_agenda_agenda"/>
      <field name="state">code</field>
      <field name="code">
        action = {
          "type": "ir.actions.act_window",
          "view_mode": "tree,form",
          "res_model": model._name,
        }
      </field>
    </record>
-->

    <!-- Top menu item -->

    <menuitem name="Mi Agenda" id="agenda.menu_root"/>

    <!-- menu categories -->

    <menuitem name="Cabecera" id="agenda.menu_1" parent="agenda.menu_root"/>

    <!-- actions -->
    <menuitem name="Lista de contactos" id="agenda.menu_1_list" parent="agenda.menu_1"
              action="agenda.action_window"/>
  </data>
</odoo>
```

Tenemos tres secciones que constituyen la vista XML:

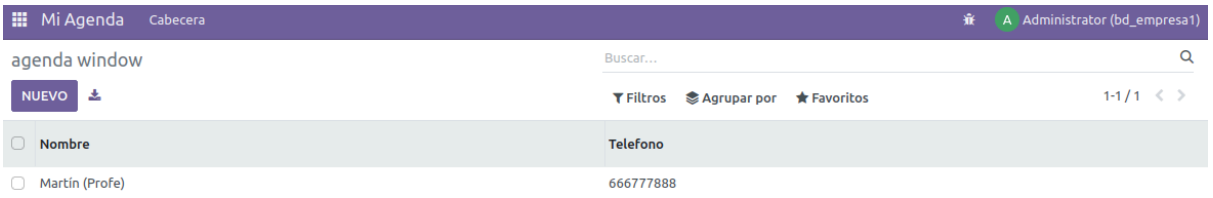
- La primera de todas corresponde a la propia vista del módulo en la aplicación, es decir, el cómo se ven los datos del módulo.
- Esta acción se relaciona con el último atributo del tercer <menuitem>. A grandes rasgos podemos concluir que en la mayoría de de módulos se enfocan en lanzar una vista concreta, en nuestro caso, va a ser la vista de inicio con los datos de nuestros contactos en la agenda.

- Tenemos también la sección de menús donde tenemos tres partes.
    - La primera corresponde al lanzador (definido así en otros módulos de configuración de odoo).
    - El segundo menú corresponde a que saldrá en la cabecera de nuestra página.
    - Submenús que pudiéramos tener.

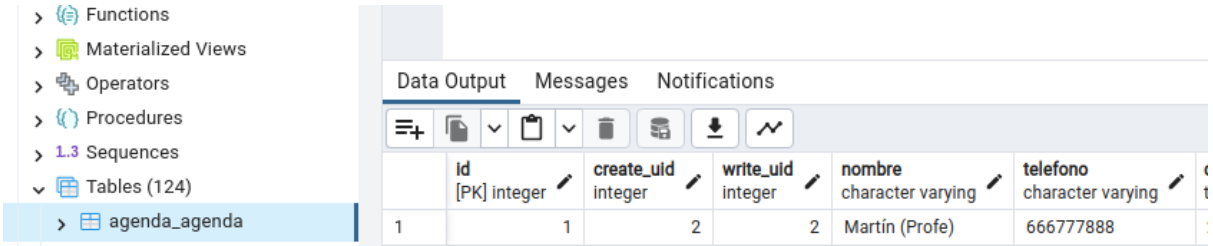
Llegados a este punto, **reinciamos Odoo y activamos nuestro módulo.**



Si te fijas la información de los menús es la misma que indicamos anteriormente. Llegados a este punto ya podemos guardar información de nuestro primer contacto y localizar en BD esta información.



Si todo ha ido bien y vamos a postgres, se nos ha creado una tabla que se llama agenda.agenda. En esta tabla está la información que vamos añadiendo, para poder verla podemos hacer click en “View/Edit data”.



5. Tarea propuesta

Tienes que crear un módulo llamado Biblioteca que va a permitir ingresar información sobre los libros que podemos tener en el IES San Clemente.

Para ello vas a tener que aportar los siguientes datos al módulo:

- Autor: Nombre del alumno.
- Descripción: Biblioteca del instituto de Formación Profesional IES San Clemente.
- URL: Página web del mestre.

Los menús tienen que tener:

- Nombre en el apartado de módulos: Biblioteca IES SC.
- Menú cabecera: Mi Biblioteca.
- Entrada del Menú Mi Biblioteca/Lista de libros.

Datos de la entidad

- Título del libro.
- Autor.
- Icbc.
- Fecha de lanzamiento.

Los datos aportados por el alumno: Todo el proceso con capturas y todos los datos, así como una captura de 5 entradas de datos en la base de datos postgres. Para que se pueda puntuar se tiene que ver en las imágenes todos los datos que anteriormente se piden.
Nota Máxima: 7/10.

Para poder optar al 8/10.
Crear un **script sql** que inserte 30 registros diferentes sobre nuestra tabla en Base de datos. Vas a tener que adjuntar el script sql con las consultas que se piden a continuación:

- **Seleccionar registros ordenados por nombre de manera ascendente.**
- **Contar el número total de registros en la tabla que comienzan por letra A en el título del libro.**
- **Contar el número total de autores diferentes que hay almacenados.**

Para poder optar al 10/10.
Investiga el módulo de contactos y crea tu propio módulo de contactos (se deberá documentar nuevamente todo el proceso). Este nuevo módulo ha de poder guardar los siguientes datos:

- Dirección con todos sus campos.
- Teléfono.
- Móvil.
- Correo electrónico.
- Identificación fiscal.
- Sitio web.
- Idioma.
- Categorías

Es importante que de hacerse estos datos tendrán que ser de un tipo apropiado para cada campo (en la página 6 se adjunta un listado de los principales datos que soporta Odoo).