IES de Teis
Avda. de Galicia, 101
36216 – Vigo

Tfno: 886 12 04 64
e-mail: ies.teis@edu.xunta.es
http://www.iesteis.es

Tarea UD4

0. Escenario

Al usar Odoo, la parte relacionada con la base de datos queda ofuscada. Esto se debe a la capa de abstracción que añade Odoo.
Crear un módulo y programarlo a bajo nivel es un punto de este curso por lo que se hace necesario dedicar al menos esta unidad didáctica a la base de datos en exclusiva.

En esta guía resolvemos la tarea propuesta para esta UD4.

Enunciado.

La tarea consiste en crear un componente o módulo que gestione una agenda telefónica con las siguientes características:

Modelo y Controlador.

- Objeto en la aplicación llamado agenda con, al menos, dos campos: Nombre y Teléfono.
- Tabla en la base de datos con los datos del objeto.

Vista.

- Menú en la aplicación que enlace al objeto.
- Vista formulario con los datos del objeto.
- Vista árbol con los datos del objeto.

Además del módulo deberás escribir también un informe con todas las consideraciones oportunas que se necesiten para entender cómo has realizado la tarea.

Criterios de puntuación. Total 10 puntos.

Los 10 puntos que tiene esta tarea están repartidos de la siguiente forma:

- 2 puntos por estructura de carpetas adecuada.
- 2 puntos por el modelo.
- 2 puntos por la vista.
- 2 puntos por los menús.
- 2 puntos por el informe.

1. Configurar carpeta para nuestros 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 portarlos a otra instancia de Odoo en otro equipo.

```
su operador_odoo
```

Con este comando cambiamos al usuario operador_odoo que es el que tiene permisos para operar en la carpeta de Odoo.

```
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 service odoo16 restart
```

Con estos dos comandos salimos del usuario operador_odoo y reinciamos Odoo para que acepte los cambios. Si recuerdas los tutoriales anteriores sabrá que también puede reiniciar el servicio de Odoo con el siguiente comando:

```
sudo systemctl restart odoo16
```

2. Crear estructura del módulo

```
su operador_odoo
```

Este comando nos cambia al usuario operador_odoo que es el único en tener permisos sobre la carpeta de Odoo.

```
/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, el andamiaje, 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. En este caso le estamos pidiendo a Odoo que cree toda la estructura de carpetas del módulo "agenda" en la nueva carpeta que hemos configurado anteriormente para nuestros módulos.

El nombre de la carpeta será el nombre técnico del módulo y debe seguir las normas python, es decir, empezar por una letra y solo contener letras, números y guion bajo.

Quedará una estructura como la siguiente:

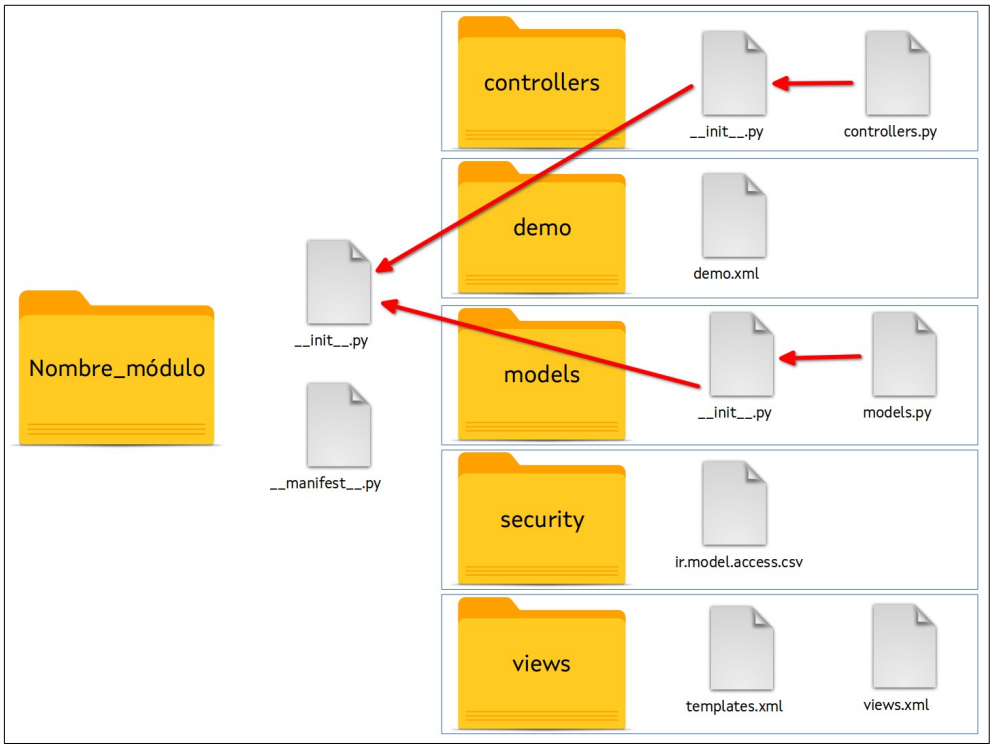
```
operador_odoo@usuario-VirtualBox:~/modulos_extra/agenda$ ls -la
total 36
drwxrwxr-x 7 operador_odoo operador_odoo 4096 mar  3 10:36 .
drwxrwxr-x 5 operador_odoo odooers       4096 mar  3 10:36 ..
drwxrwxr-x 2 operador_odoo operador_odoo 4096 mar  3 10:36 controllers
drwxrwxr-x 2 operador_odoo operador_odoo 4096 mar  3 10:36 demo
-rw-rw-r-- 1 operador_odoo operador_odoo   71 mar  3 10:36 __init__.py
-rw-rw-r-- 1 operador_odoo operador_odoo  913 mar  3 10:36 __manifest__.py
drwxrwxr-x 2 operador_odoo operador_odoo 4096 mar  3 10:36 models
drwxrwxr-x 2 operador_odoo operador_odoo 4096 mar  3 10:36 security
drwxrwxr-x 2 operador_odoo operador_odoo 4096 mar  3 10:36 views
operador_odoo@usuario-VirtualBox:~/modulos_extra/agenda$
```

3. Entendiendo la estructura de carpetas del módulo

Un módulo debe contener, como mínimo dos archivos: el **__init__.py** que enlaza con otros archivos y **__manifest__.py** que describe el módulo con al menos el campo de nombre.

El archivo **__init__.py** es típico en proyectos python y sirve para unificar un conjunto de paquetes o archivos del proyecto especialmente si están distribuidos en varias carpetas. De esta manera, importando un único __init__.py podemos enlazar todos los paquetes y clases que tengamos en distintos archivos.

En la estructura de carpetas que se crea al usar **scaffold** estos son los archivos __init__py. Si te fijas, en el primero, el que está en la raíz del proyecto, se referencian los otros archivos __init__.py que están en las subcarpetas controllers y models (que es donde actualmente hay otros archivos python). A su vez estos archivos __init__.py referencian a los archivos python que se encuentran en esa subcarpeta.



Si en algún momento añadimos otro archivo python a alguna subcarpeta será suficiente con importarlo en el __init__.py de esa subcarpeta para que esté incluido y accesible a todo el proyecto

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.

Este archivo se describe de la siguiente manera:

```
# -*- coding: utf-8 -*-
{
    'name':"agenda",

    'summary':"""
    Módulo de la UD4.
    """,

    'description':"""
    Módulo encargado de almacenar una agenda de contactos.
    """,

    'author':"Javi",

    'website':"www.edu.xunta.gal/centros/iesteis",

    'category':'Uncategorized',

    'version':'0.0.1',
```

```
    'depends':['base'],

    'data':[
        'security/ir.model.access.csv',  #archivo de permisos
        'views/views.xml',               #archivo de vistas, acciones y menús
    ],
}
```

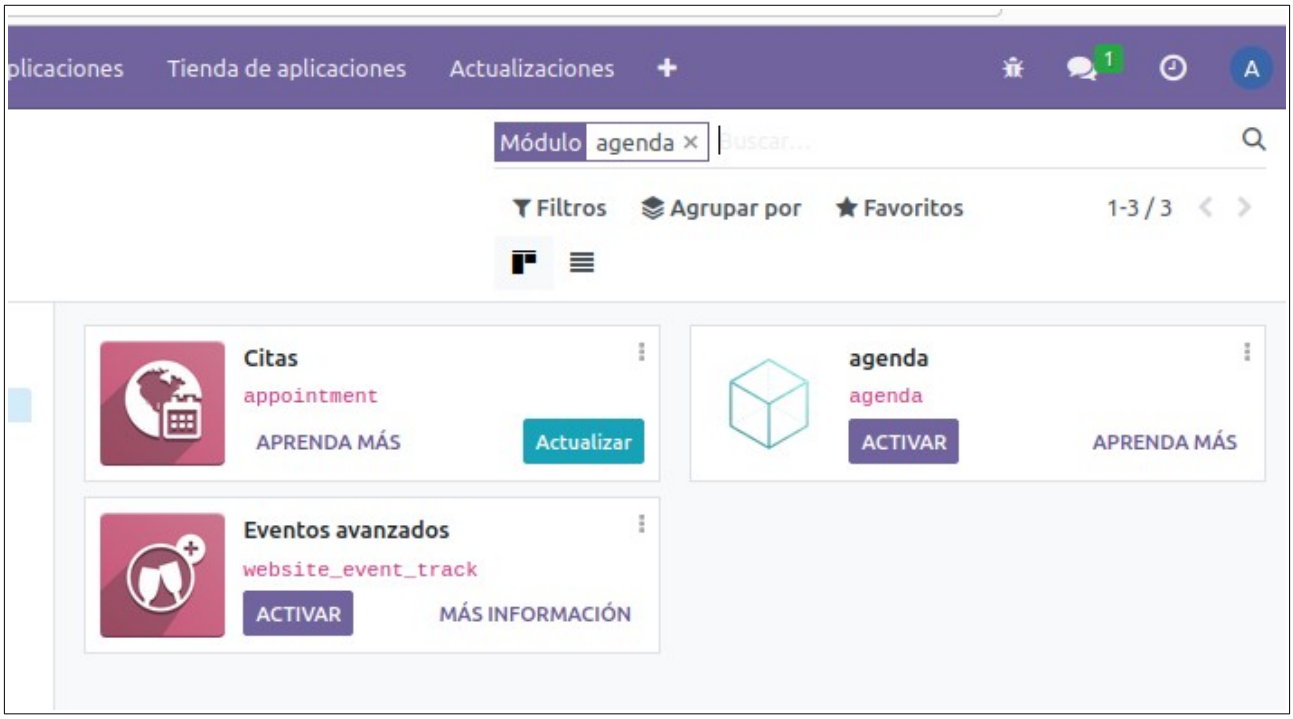
Existen mas campos que se pueden consultar desde la documentación oficial de Odoo pero, en general no es necesario un nivel de detalle alto salvo que pretendamos publicar el módulo en la tienda de Odoo. Estos son:

- La clave **name** guarda el título del módulo.
- La clave **description** es una descripción del funcionamiento del módulo.
- En **author** se guarda el nombre del creador.
- La clave **depends** es una lista de los módulo que deben ser instalados para que funcione el nuestro.
- En **application** indicamos si el módulo tiene entidad suficiente para aparecer en las aplicaciones o es una ampliación de otros módulos y no requiere entrada en aplicaciones.
- La clave **summary** es un subtítulo del módulo.
- En **version** guardamos la iteración en el desarrollo del módulo.
- La clave **license** indica el tipo de licencia del módulo creado.
- En **website** se guarda la URL de la web con más información del módulo.
- La clave **category** indica la categoría a la que pertenece el módulo. Si no existe se crea.
- En **installable** indicamos si el módulo es instalable. Por defecto es verdadero.
- La clave **auto_install** a verdadero indica que este módulo debe instalarse solo si en cuanto se cumplan las dependencias que tenga.
- La clave **data** es una lista de todos las vistas que se usan en esta módulo.

Recuerda que después de crear el archivo __manifest__py dentro de alguna de las carpetas designadas para almacenar addons es necesario reiniciar Odoo. Podemos hacerlo de varias maneras pero la más sencilla es por comando.

```
sudo systemctl restart odoo16
```

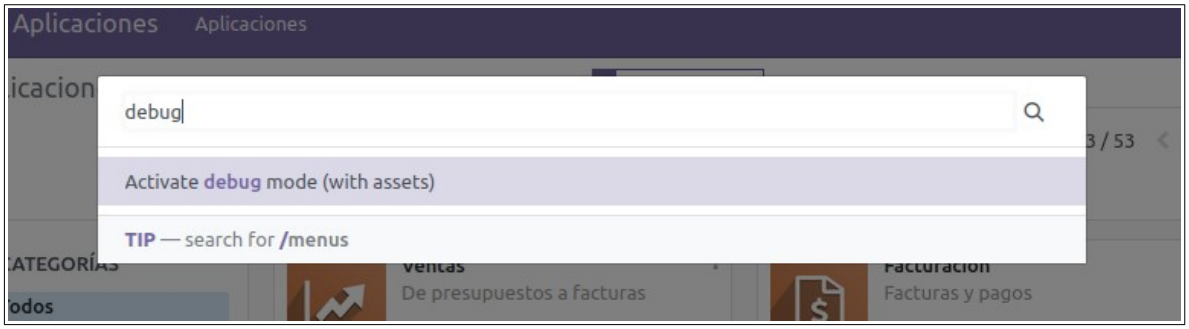
Con este comando reiniciamos Odoo y le obligamos a revisar todas las carpetas de Odoo para actualizar los cambios. Para que se carguen estos cambios será necesario actualizar explícitamente las aplicaciones de Odoo y actualizar el módulo.



4. Activar el modo desarrollo en la interfaz web de Odoo

Odoo ofrece un modo desarrollador en el que podemos ver las opciones extendidas que nos servirán para programar y personalizar Odoo. Para un usuario normal que se dedique a trabajar con Odoo no es necesaria esta vista.

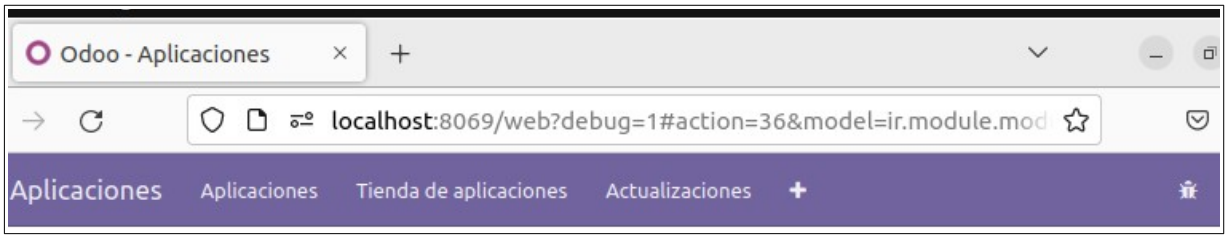
La forma más sencilla de activar y desactivar el modo desarrollo consiste en abrir la paleta de comandos mediante la combinación de teclas **Ctrl + K** y escribir *debug*.



Sabrás que tienes el modo desarrollador activado por la cantidad de opciones que ves y también porque los parámetros de la URL en tu navegador comienza por "debug".

De la misma manera podremos desactivar este modo cuando nos interese.

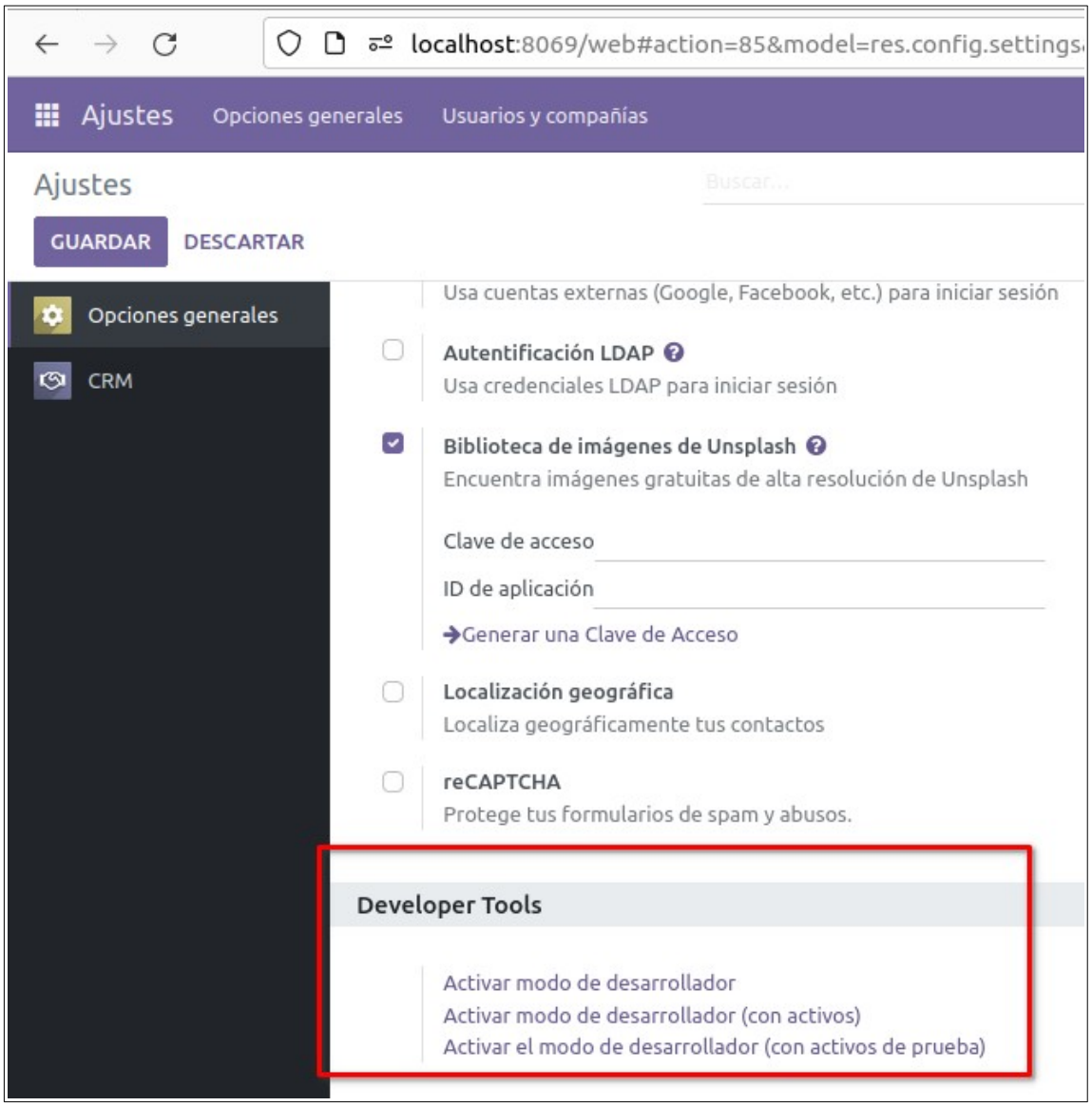
Activa y desactiva este modo en diferentes pantallas para ver cuales son las opciones y menús extra que aporta.
Otra manera de activar el modo desarrollador es precisamente la comentada hace un momento, modificar la URL escribiendo **?debug=1** justo después del */web* de la URL



Para desactivar el modo desarrollador en este caso podemos eliminar el debug o ponerle valor 0.

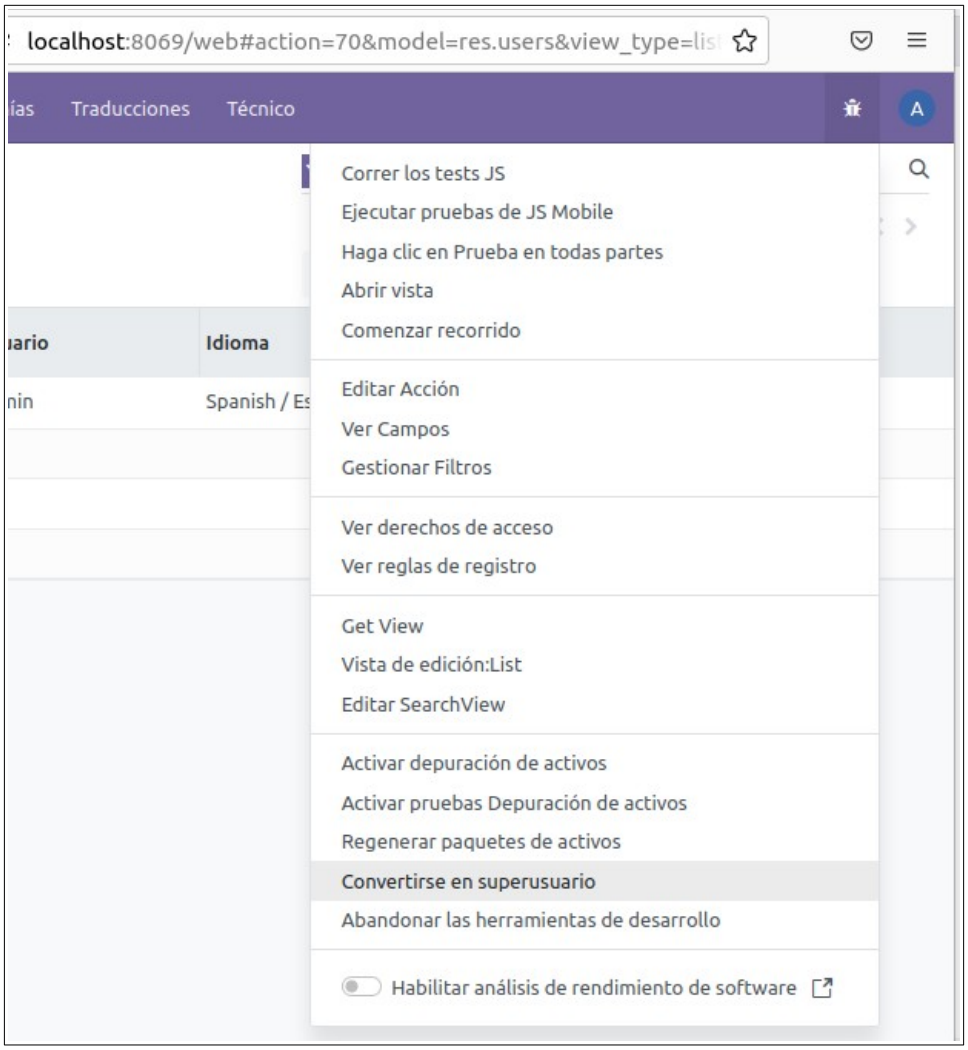
Otra forma para activar este modo es instalar una extensión llamada "Odoo Debug" en el navegador.

La última forma de activar el modo desarrollador es mediante la interfaz web de Odoo pero para ello es necesario tener al menos una aplicación instalada y por ahora nosotros no tenemos ninguna (es una instalación limpia). En ese momento podremos activar y desactivar el modo desarrollador en Ajustes / Opciones Generales / Developer Tools y allí activar el modo desarrollo.



Recuerda que esta opción no aparece mientras no tengas al menos una aplicación instalada.

Con el modo desarrollador activo disponemos de un menú en la esquina superior derecha con herramientas propias para ver el código y realizar cambios.



5.- Los modelos

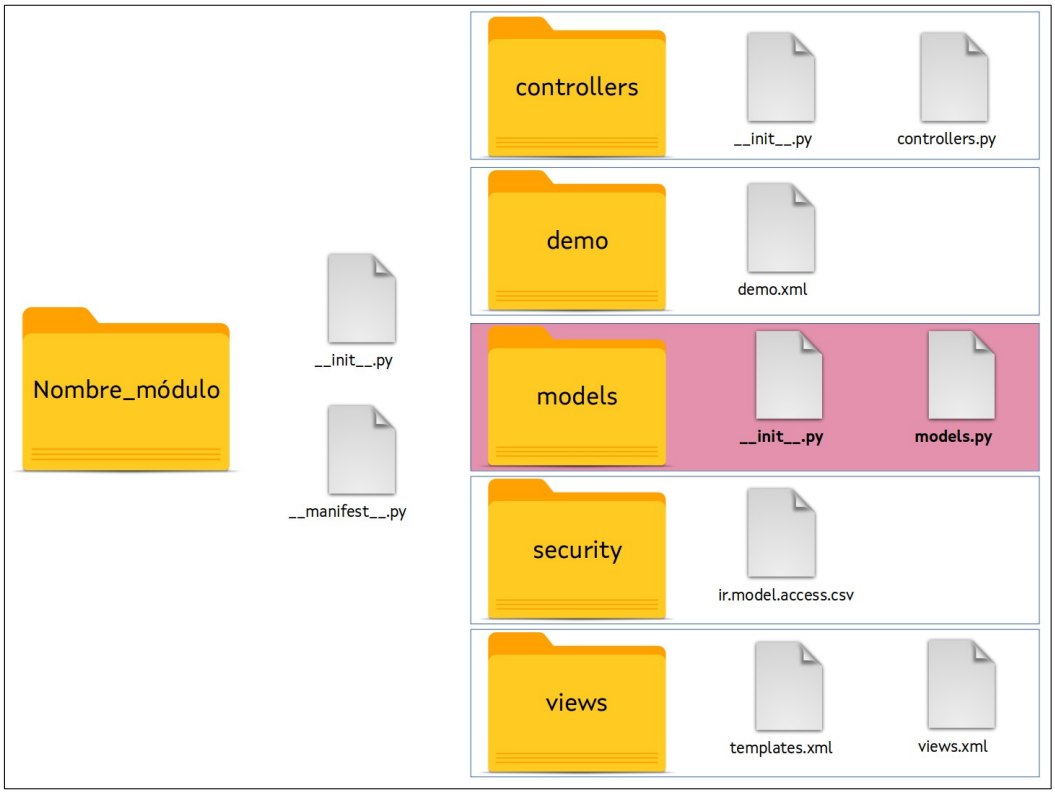
No es estrictamente necesario programar en python usando la capa ORM, podríamos ejecutar sentencias SQL directamente sobre la base de datos pero nos estaríamos saltando todas las validaciones de Odoo. En general se considera mucho más eficiente usar la API ORM de Odoo ya que proporciona mecanismos de seguridad y autenticación contra la base de datos.

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.

```
#Archivo ejemplo_modelo.py

from odoo import models

class EjemploModelo(models.Model):
```

```
    _name = “ejemplo.model”
    _description = “Texto largo de descripción”
```

Al heredar de la clase Model se generará automáticamente una tabla en la base de datos que tomará el nombre de _name y sustituirá el punto por un guión bajo. En este caso creando la tabla “ejemplo_model”.



Los modelos deberían estar guardados en la carpeta **models** de la estructura de subcarpetas del módulo y es importante actualizar en archivo **__init__.py** de la subcarpeta models para incluir este nuevo archivo con la clase EjemploModelo.

```
#Archivo __init__.py

from . import models
```

Ademas de todos los atributos y métodos que puedan heredar también podemos añadir nuestros propios atributos e indicar el tipo de datos que gestionan. Algunos de ellos almacenan valores atómicos en la base de datos, otros enlazan registros de unas tablas con otras. Los campos y parámetros de Odoo requieren importar la clase fields

```
from odoo import models, fields
```

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 los parámetros necesarios.

La tarea de esta unidad didáctica pide crear un nuevo módulo llamado "Empresa" con una serie de campos que principalmente tienen su origen en la tabla res_partner. Recuerda que no todos, el campo credit_limit está en otra tabla. Consulta la guía de consultas a la base de datos si necesitas recordarlo.

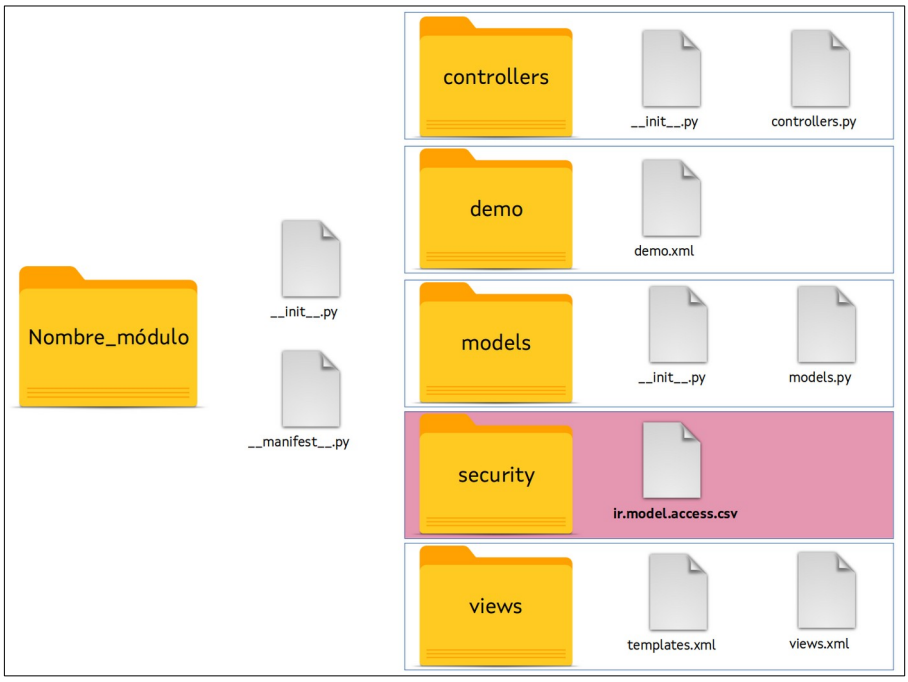
Por tanto en este módulo no vamos a crear ninguna tabla extra en la base de datos, vamos a usar res_partner. Esta es la razón por la que no indicamos ningún _name ni ningún otro campo a añadir en esa tabla.

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

from odoo import models, fields, api

class agenda(model.Model):
     _name = "agenda.agenda"
     _description = "Modelo agenda UD4"

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

En este código del archivo **model/models.py** creamos una clase llamada agenda. La propiedad _name generará una nueva entrada en la base de datos en forma de nueva tabla. Las columnas de esa tabla serán nombre y teléfono ambas de tipo char.

6. Seguridad

El mecanismo de seguridad que incluyen los módulos debe indicar a qué grupos de usuarios se les permite acceder a los datos.
Cuando los permisos de acceso sobre un modelo no están definidos ningún usuario puede cargarlos.
A nivel interno Odoo guarda los permisos en registros de la **table ir.model.access** con información del modelo, grupo y conjunto de premisos (crear, leer, escribir y borrar)
Dentro de la estructura de carpetas del módulo, el archivo donde se define las reglas de seguridad es el achivo **ir.model.access.csv** en la carpeta **security**.



El archivo i**r.model.access.csv**

```
id,name,model_id:id,group_id:id,perm_read,perm_write,perm_create,perm_unlink
agenda_regla1,regla1 agenda,model_agenda_agenda,base.group_user,1,1,1,1
```

Con estas líneas estamos generando una nueva regla de acceso para el modelo donde todo el mundo tiene todos los permisos.

El campo **id** indica el identificador externo, aquel que identifica al registro de manera unívoca en la base de datos sin tener que conocer el verdadero id interno que tiene en postgres. Usaremos alguna nomenclatura significativa que nos ayude a conocer qué hace esa regla y sobre qué modelo.

El campo **name** es igual al id pero será lo que se muestre al usuario en la interfaz gráfica. Debe ser un texto único y significativo para el permiso.

El campo **model_id/id** identifica al modelo sobre el que aplicamos los permisos. El nombre correcto debería ser "**model_**" y el **_name** del modelo por lo que es posible que debas volver a la carpeta models y consultar esa información en el archivo python.

El campo **group_id/id** indica el grupo al que se le aplica los derechos de acceso. Odoo entiende que si el grupo se deja vacío es porque queremos que los permisos se apliquen a todos los usuarios. Se puede definir nuevos grupos en archivos XML, usar los de Odoo de la tabla **res.groups** o usar **base.group_user** para referirnos a cualquier usuario.

Los campos **perm_read, perm_write, perm_create y perm_unlink** indican lectura, escritura, creación y borrado permitidos con un 1 y denegaos con un 0.

Además de la línea de cabecera tendremos tantas otras como permisos a grupos distintos queramos dar. Recuerda cambiar las ids porque si pones la misma estarías sobrescribiendo la misma regla una y otra vez. Si no solo estuviera la cabecera sin ninguna línea más indicaría que solo el superusuario tiene permisos.

Los permisos son aditivos. Si un usuario tiene permiso de lectura por pertenecer a un grupo y permiso de escritura por pertenecer a otro grupo finalmente tiene ambos permisos, no se excluyen, se suman.

7. Vistas

Las vistas se definen en archivos XML junto con las acciones y menús. Son instancias del model ir.ui.view

```
<record id="MODEL_view_TYPE" model="ir.ui.view">
  <field name="name">NAME</field>
  <field name="model">MODEL</field>
  <field name="arch" type="xml">
    <VIEW_TYPE>
      <VIEW_SPECIFICATIONS/>
    </VIEW_TYPE>
  </field>
</record>
```

name es un nombre descriptor de la vista.

model el el modelo al que se refiere esta vista

arch es la descripción del diseño de la vista

A un nivel básico las vistas disponibles para el VIEW_TYPE son de estos 2 tipos:

List. Su elemento es <tree> y cada campo se corresponde con una columna. Esta vista muestra los registros en forma de tabla o lista. Es una vista que muestra varios registros.

```
<tree string="Lista de la compra">
    <field name="elemento"/>
    <field name="cantidad"/>
</tree>
```

Form. Su elemento es <form> y estan compuestos de estructuras de alto nivel (groups y notebooks) y de elementos interactivos (buttons y fields). Esta vista muestra un registro en detalle.

```
<form string="Test">
    <sheet>
        <group>
            <group>
                <field name="name"/>
            </group>
            <group>
                <field name="last_seen"/>
            </group>
            <notebook>
                <page string="Description">
                    <field name="description"/>
                </page>
            </notebook>
        </group>
    </sheet>
</form>
```

En el apartado 10 pongo un ejemplo de las vistas enfocado a la tarea de esta unidad pero antes un ejemplo de como quedaría un archivo views.xml con una vista, una acción y un menú.

Tanto las vistas, las acciones y los menús pueden declararse en el mismo archivo. Las etiquetas <record model="tabla"> son nuevos registros que se insertarán en la tabla "tabla". También podríamos declarar cada elemento en un archivo distinto pero en ese caso es necesario declararlo en la sección "data" del archivo __manifest__.py en la raíz del módulo.

Lo que sí es necesario, tanto si se declaran juntos como si se declaran en archivos distintos, es que estén en este orden ya que unas secciones dependen de otras. Fíjate que en la parte de las acciones se hace re-

ferencia a un id de vista (verde). Por otro lado, en la sección de menús se hace referencia a un id de acción (rojo).

8. Acciones

Las acciones son registros en una tabla de la base de datos. Para añadirlos podemos usar archivos CSV que son los preferidos para importaciones masivas de datos simples por su velocidad y sencillez. También podemos usar archivos XML que son más fáciles de leer y estructurar aunque menos eficientes.

Normalmente la interfaz de Odoo se basa en el patrón Menú-Acción-Vista donde el usuario navega a través de niveles de menús para llegar a una acción en la que se muestran uno o varios registros en una vista.

Las acciones se pueden disparar mediante click en los elementos de los menús o bien mediante click en botones de las vistas.

Generalmente definimos las acciones mediante un archivo XML almacenado en la carpeta views. Este archivo es necesario incluirlo en la sección 'data' del archivo __manifest__.py de la raíz de las carpetas del módulo.

Una acción básica para el mi.modelo podría ser:

```
<record id="mi_modelo_action" model="ir.actions.act_window">
    <field name="name">Test action</field>
    <field name="res_model">test.model</field>
    <field name="view_mode">tree,form</field>
    <field name="view_id" ref="id_de_la_vista"/>
</record>
```

- id es el identificador externo que será usado para referenciar ese registro sin necesidad de conocer su id en la base de datos postgres. Se usa como separador el guión bajo.

- model es un valor fijo ir.actions.act_window

- name es el nombre de la acción

- res_model es el modelo a la que aplicamos la acción. Debe coincidir con el nombre _name definido en el modelo.

- view_mode son las vistas (y orden de ellas) disponibles

-view_id se corresponde con el id de la vista definido anteriormente y será la vista mostrada al ejecutar esta acción.

9. Menús

Los menús son registros que se guardan en **ir.ui.menu**. Para declararlos usamos <menuitem>

<menuitem id="test_model_menu_action" action="test_model_action"/>

Tenemos menús en 3 niveles. El primero se corresponde con la vista principal de las aplicaciones, el segundo nivel es la barra superior y el tercero es el menú de acción.

```
<menuitem id="test_menu_root" name="Test">
    <menuitem id="test_first_level_menu" name="First Level">
        <menuitem id="test_model_menu_action" action="test_model_action"/>
    </menuitem>
</menuitem>
```

En este código hay anidados 3 niveles de menús. El primero se corresponde con la vista en el menú de aplicaciones. El segundo

En algún menuitem pondremos un parámetro action junto con el id de la acción que queremos desencadenar cuando el usuario pinche en ella.

10. Código fuente.

Contenido del archivo views.xml

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

    <record model="ir.ui.view" id="agenda.list">
      <field name="name">Listado de agenda</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">Acción de la ventana de agenda</field>
      <field name="res_model">agenda.agenda</field>
      <field name="view_mode">tree,form</field>
    </record>


    <!-- Top menu item -->

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

    <!-- menu categories -->

    <menuitem name="Menu 1" id="agenda.menu_1" parent="agenda.menu_root"/>
```

```
    <!-- actions -->

    <menuitem name="List" id="agenda.menu_1_list" parent="agenda.menu_1"
              action="agenda.action_window"/>

  </data>
</odoo>
```

Contenido del archivo models.py

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

from odoo import models, fields, api


class agenda(models.Model):
    _name = 'agenda.agenda'
    _description = 'Modelo agenda UD4'

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

Contenido del archivo ir.model.access.csv

```
id,name,model_id:id,group_id:id,perm_read,perm_write,perm_create,perm_unlink
agenda_regla1,regla1 agenda,model_agenda_agenda,base.group_user,1,1,1,1
```

Contenido de __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': """
        Long description of module's purpose
    """,

    'author': "My Company",
    '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_module_categor
```

IES de Teis
Avda. de Galicia, 101
36216 – Vigo
Tfno: 886 12 04 64
e-mail: ies.teis@edu.xunta.es
http://www.iesteis.es


```
y_data.xml
    # 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',
    ],
}
```