UT 8. Desarrollo Odoo y Python.

Prof. D. Rafael Reina

Sistema de Gestión Empresarial

Contenidos

1. Lenguaje Python.
2. Entorno Python.
3. Estructura de un módulo base en Odoo
4. Integración de aplicaciones empresariales.
5. Desarrollo de Microservicios con Spring.
6. Entorno Spring Boot.

RA y Objetivos

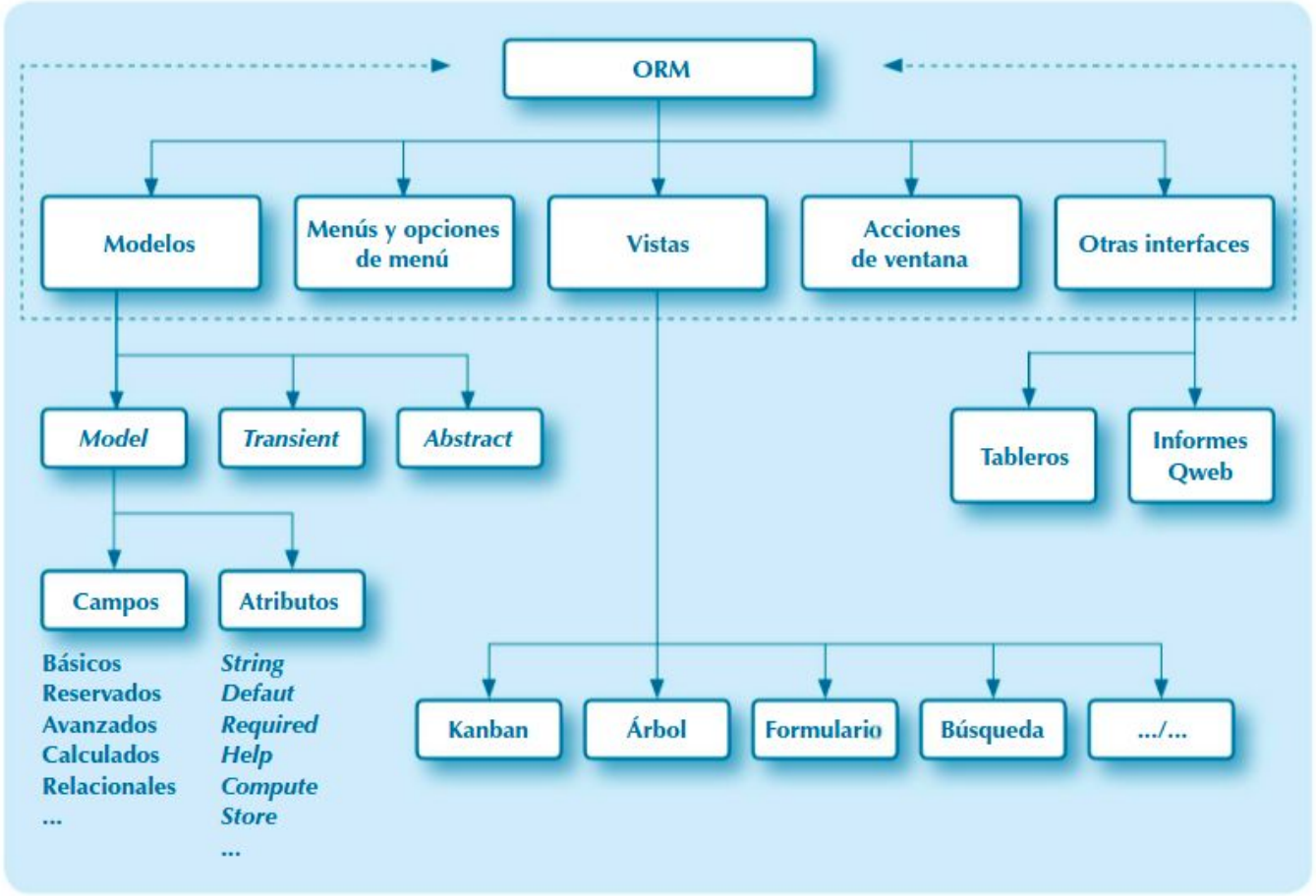
UT 5. Desarrollo de nuevos componentes para ERP-CRM. 20 horas FEB	RA 5. Desarrolla componentes para un sistema ERP-CRM analizando y utilizando el lenguaje de programación incorporado.	q) Seleccionar y emplear lenguajes y herramientas, atendiendo a los requerimientos, para desarrollar componentes personalizados en sistemas ERP-CRM. w) Identificar los cambios tecnológicos, organizativos, económicos y laborales en su actividad, analizando sus implicaciones en el ámbito de trabajo, para mantener el espíritu de innovación.

TAREAS Planificadas

TAREAS RA 4	HS	Entregables	Criterios de evaluación	Instrumentos	Criterios
1. Definir un modelo de negocio. 2. Realizar el análisis de las necesidades de automatización mediante historias de usuario.	2	Tarea 8. Definir requisitos de desarrollo en Odoo.		Planificación de las tareas para el desarrollo ágil(70 %) Informe del alumno (10 %) Defensa práctica (20%)	30 %
1. Analizar las necesidades de desarrollo en Odoo.	4		a) Se han reconocido las sentencias del lenguaje propio del sistema ERP-CRM. b) Se han utilizado los elementos de programación del lenguaje para crear componentes de manipulación de datos.		
1. Concretar una metodología de desarrollo ágil.	2	Tarea 9. Aplicando Scrum planificar y desarrollar una API Rest y consumirla con Odoo.	c) Se han modificado componentes software para añadir nuevas funcionalidades al sistema.		
1. Planificar las las fases de desarrollo. 2. Planificar las tareas de desarrollo por cada fase.	4		f) Se han documentado todos los componentes creados o modificados.		
1. Desarrollar soluciones de integración mediante API REST.	8		d) Se han integrado los nuevos componentes software en el sistema ERP-CRM. e) Se ha verificado el correcto funcionamiento de los componentes creados.		
	20				

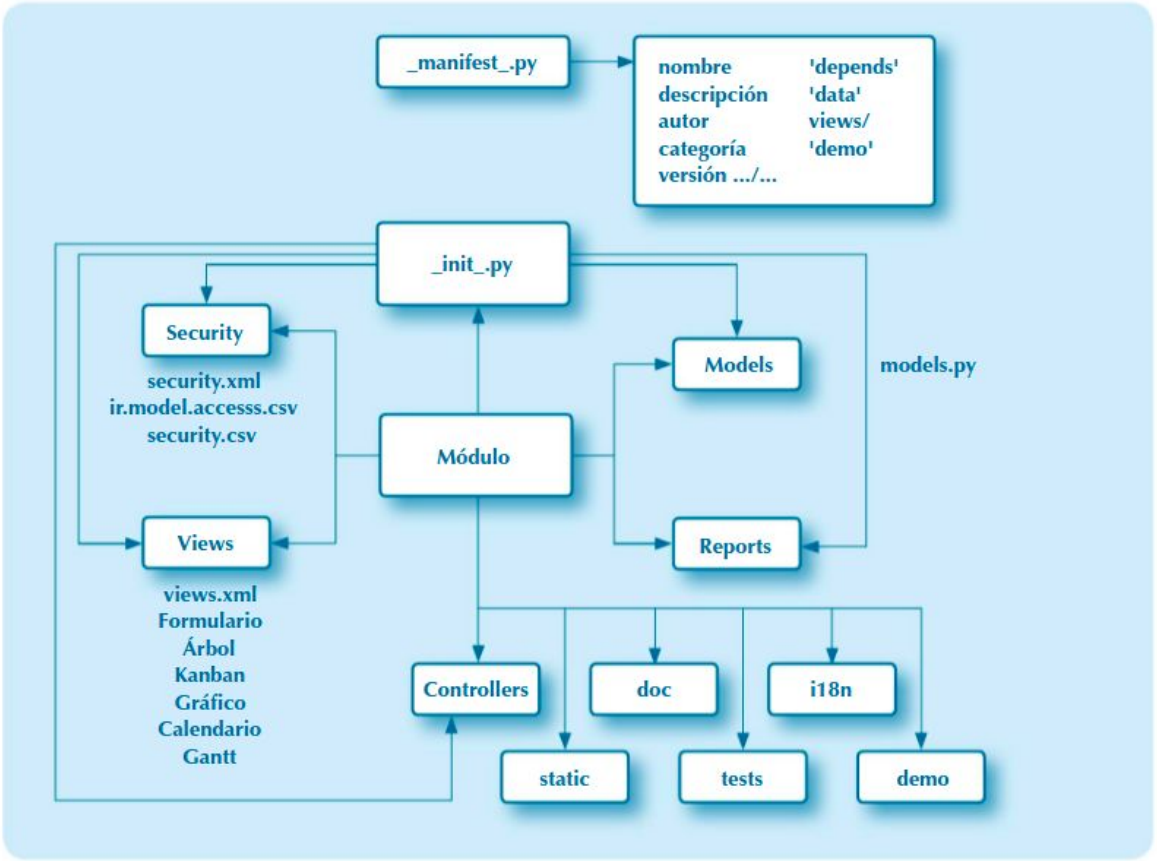
Visión general de Odoo

Mapa conceptual



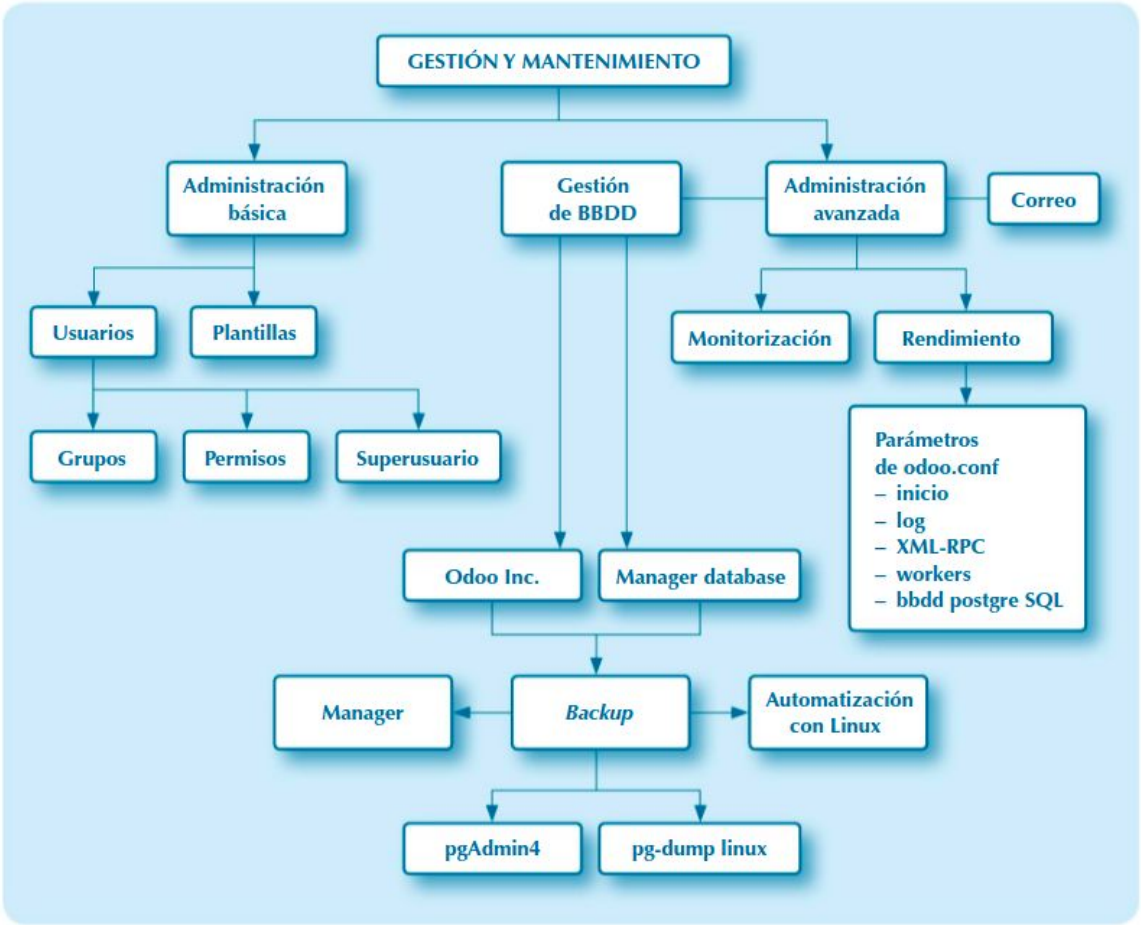
Estructura módulo en Odoo

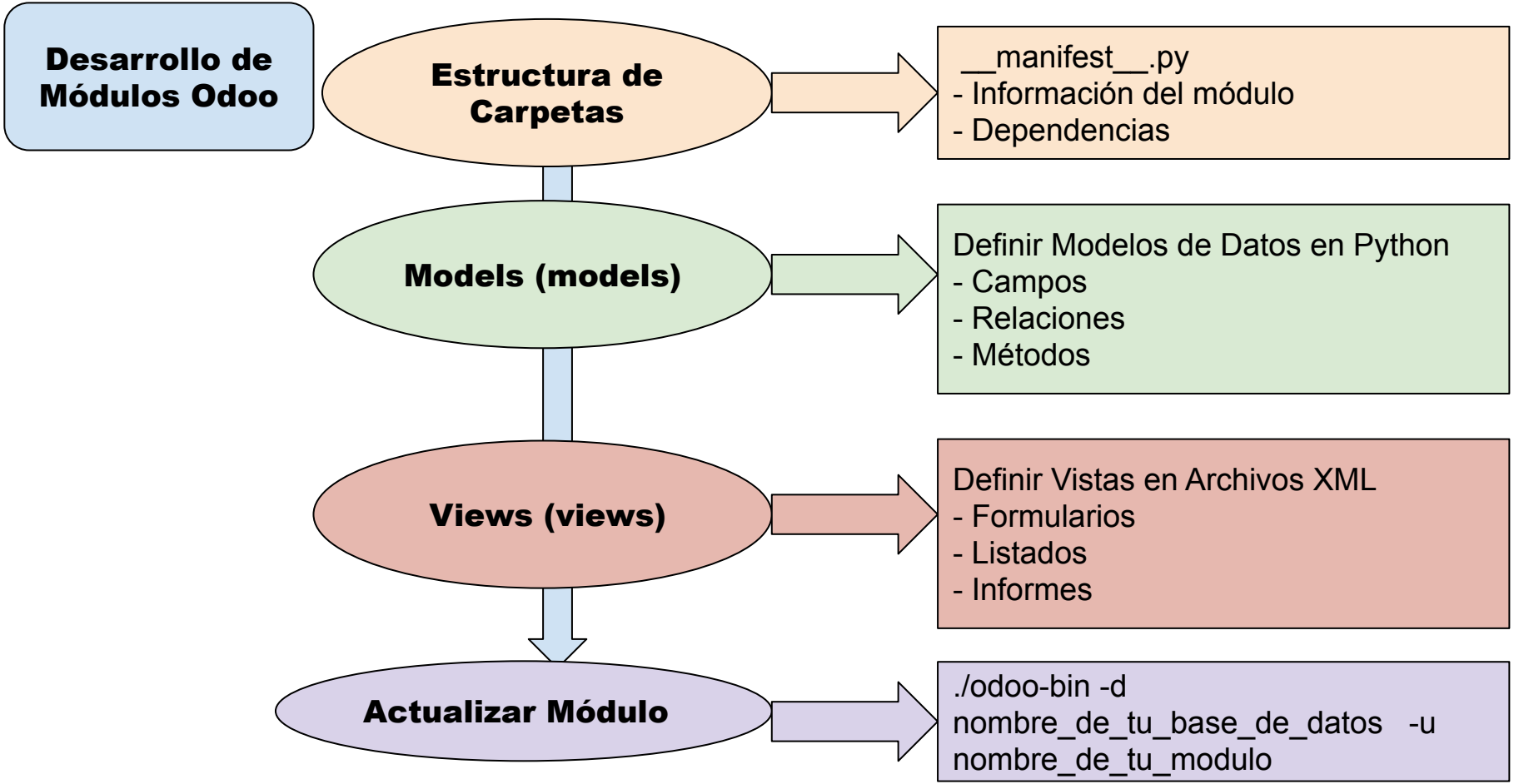
Mapa conceptual



Gestión y mantenimiento en Odoo

Mapa conceptual

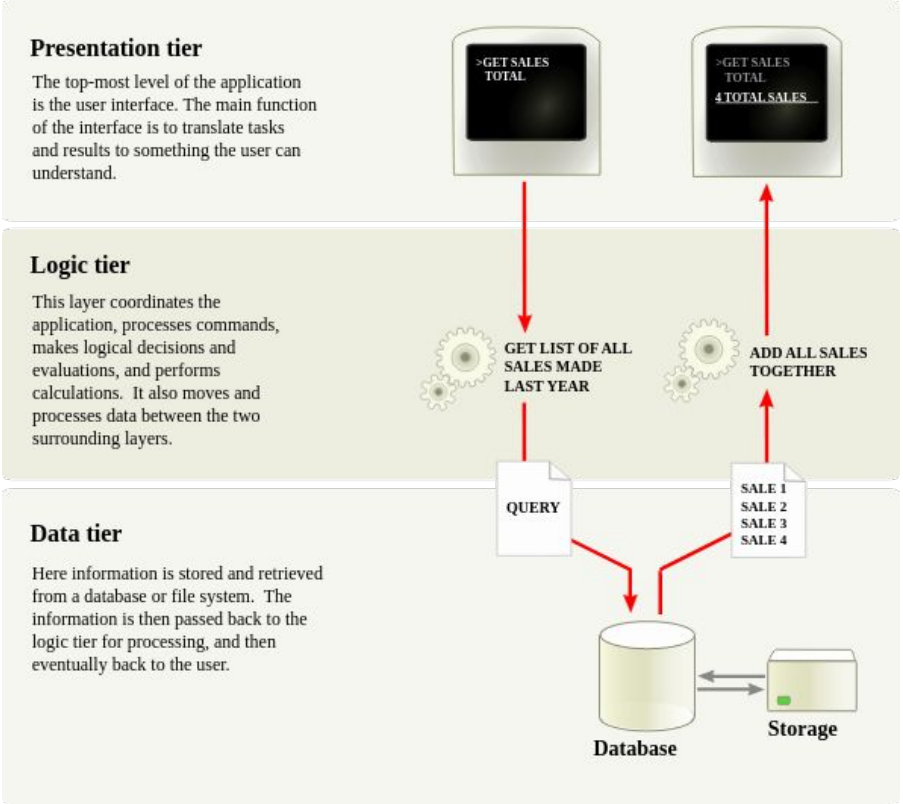



Desarrollo de Módulos Odoo
Estructura de Carpetas
__manifest__.py
- Información del módulo
- Dependencias
Models (models)
Definir Modelos de Datos en Python
- Campos
- Relaciones
- Métodos
Views (views)
Definir Vistas en Archivos XML
- Formularios
- Listados
- Informes
Actualizar Módulo
./odoo-bin -d nombre_de_tu_base_de_datos -u nombre_de_tu_modulo

ARQUITECTURA módulo

Arquitectura multicapa.

```
module
├── models
│   ├── *.py
│   └── __init__.py
├── data
│   └── *.xml
├── __init__.py
└── __manifest__.py
```



Localización y estructura de un módulo en ODOO

En la carpeta “addons” se encuentran los módulos básicos disponibles de Odoo que se pueden instalar.

Cada a”addons” tiene la siguiente estructura de subdirectorios básica:

- __manifest__.py : metadatos del módulo y definición de ficheros XML que deben cargarse
- __init__.py : archivo principal del módulo y carga de código Python.
- models: definición de objetos ORM en Python.
- views: ficheros XML con la definición de las GUIs (vistas, acciones, elementos de menú)
- controllers : archivos de código para controladores y rutas http de sitios web.

Estructura de un módulo en ODOO

Otros subdirectorios del addons son:

- data: datos de ejemplo a cargar en la base de datos (xml o css).
- report: modelos de informes en formato XML.
- security: archivos xml y csv donde se especifican los permisos de grupos y reglas de seguridad (listas de control de acceso). Suelen contener los archivos *ir.models.access.csv y security.xml*
- static: documentación del sitio web 8css, js, lib,...)
- i18n: archivos de localización y traducción.
- wizard: vistas tipo wizard. Reagrupa modelos transitorios.
- tests: código para pruebas de funcionamiento.

Entorno de desarrollo

IDE: VSCodem Sublime, Text, Atom, PyCharm.

https://www.odoo.com/documentation/15.0/es/developer/tutorials/getting_started/02_setup.html

git clone https://github.com/odoo/odoo.git

Creación de un módulo versión 1

https://www.odoo.com/documentation/14.0/howtos/backend.html

Esquema de un módulo vacío:

sudo python 3 /opt/odoo/odoo/odoo-bin **scaffold** mi_primerModulo /opt/odoo/odoo/addons

https://www.odoo.com/documentation/14.0/es/developer/howtos/backend.html?highlight=scaffold

Estructura básica con SCAFFOLD

```
total 28

drwxr-xr-x 2 root root 4096 ene 16 17:47 controllers
drwxr-xr-x 2 root root 4096 ene 16 17:47 demo
-rw-r--r-- 1 root root   71 ene 16 17:47 __init__.py
-rw-r--r-- 1 root root  914 ene 16 17:47 __manifest__.py
drwxr-xr-x 2 root root 4096 ene 16 17:47 models
drwxr-xr-x 2 root root 4096 ene 16 17:47 security
drwxr-xr-x 2 root root 4096 ene 16 17:47 views
```

Creación módulo versión 2

Herramientas para automatizar la creación de un módulo simple y con herencia:

https://github.com/falconsoft3d/odoo-constructor-module

Nota Windows: La vista enlazada desde el __manifest__.py debe tener como nombre “views.xml” para que sea visible al actualizar las aplicaciones en Odoo.

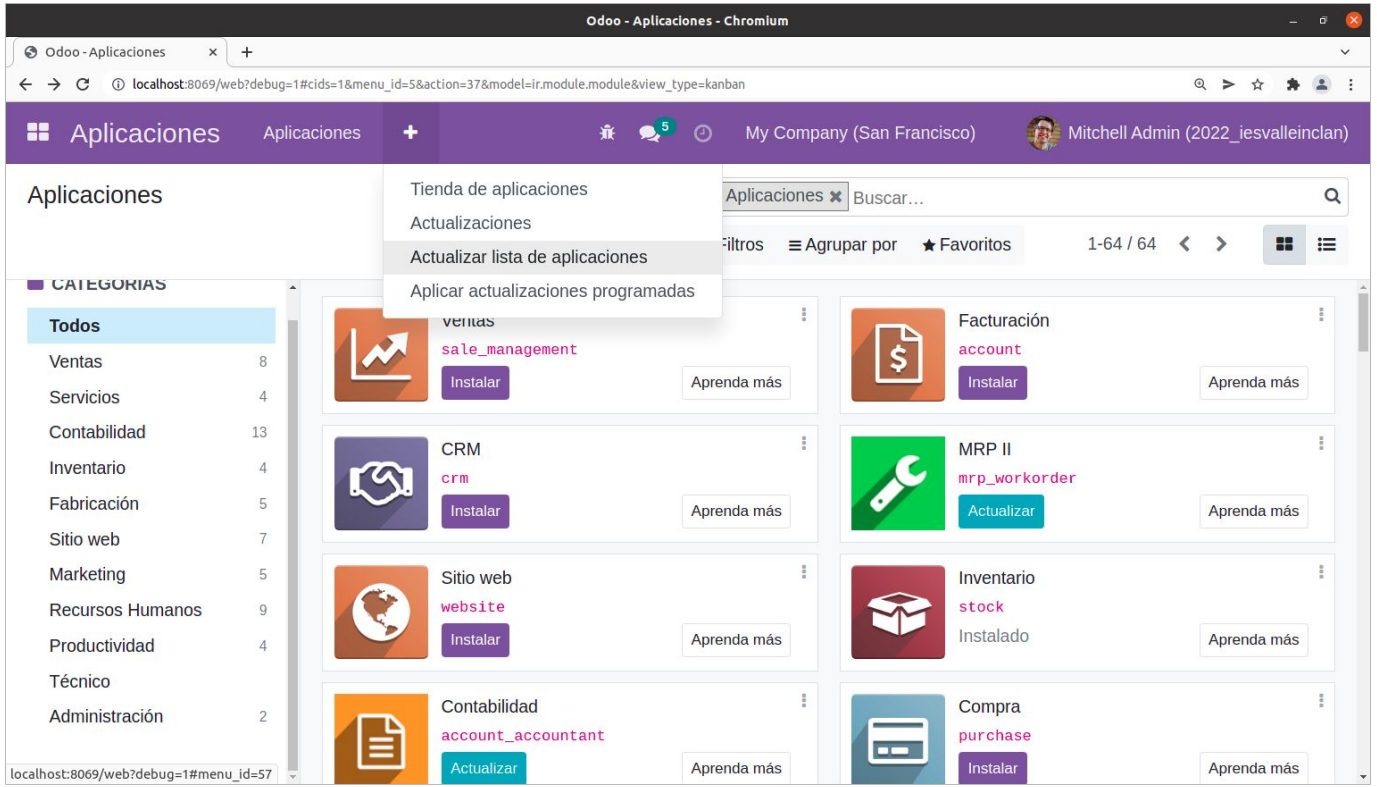
Python3 consideraciones

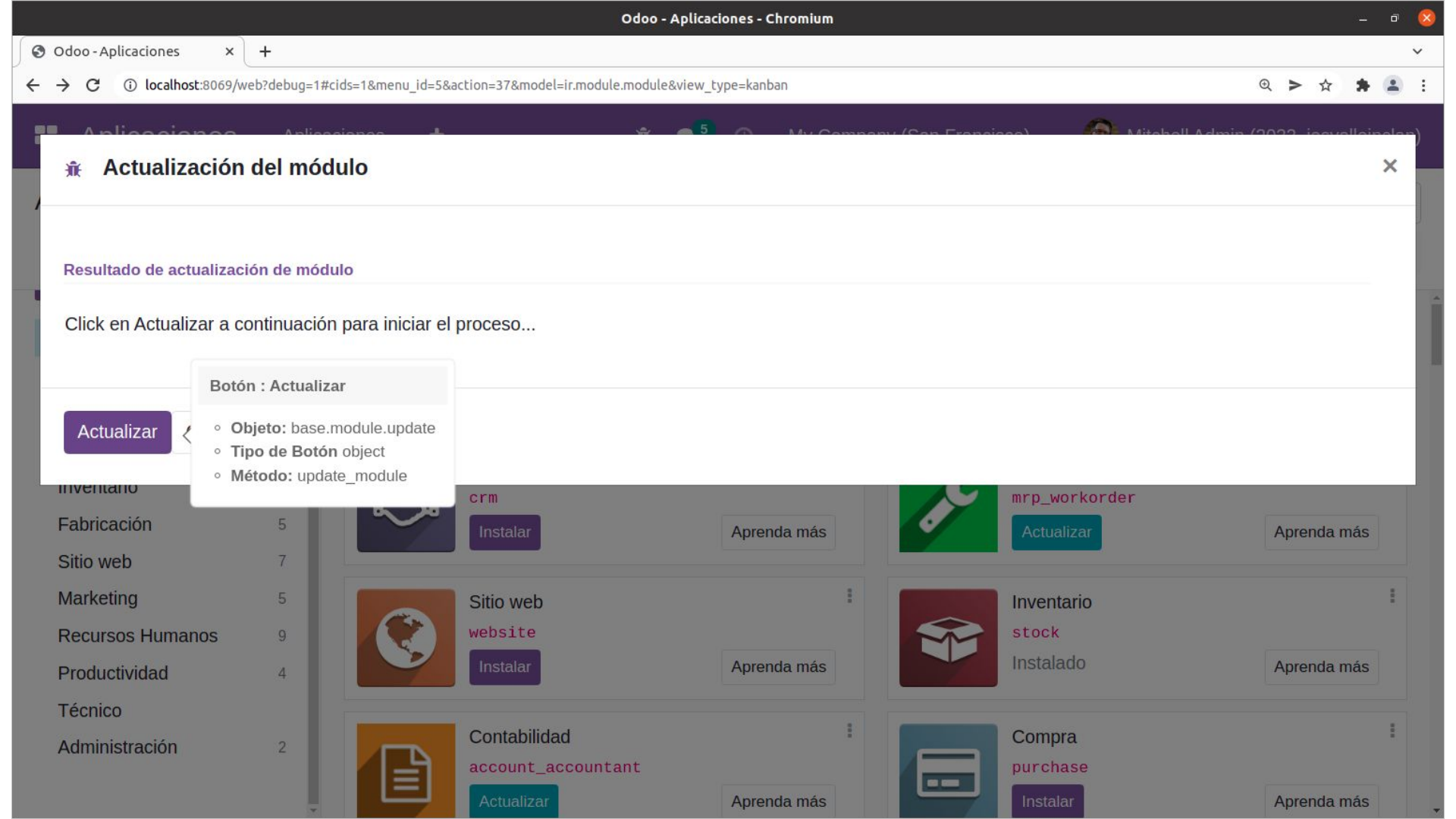
En el script hay que sustituir el método raw_input por input.

El nombre del módulo debe comenzar por minúscula para que pueda ser reconocido por ODOO.

Se debe copiar el módulo generado en la carpeta addons y actualizar desde Odoo la lista de aplicaciones.

Actualizar lista de aplicaciones en modo desarrollador




Odoo - Aplicaciones - Chromium
Odoo - Aplicaciones
localhost:8069/web?debug=1#cids=1&menu_id=5&action=37&model=ir.module.module&view_type=kanban
Actualización del módulo
Resultado de actualización de módulo
Click en Actualizar a continuación para iniciar el proceso...
Botón : Actualizar
Objeto: base.module.update
Tipo de Botón object
Método: update_module
Actualizar
Fabricación 5
Sitio web 7
Marketing 5
Recursos Humanos 9
Productividad 4
Técnico
Administración 2
crm
Instalar
Aprenda más
mrp_workorder
Actualizar
Aprenda más
Sitio web
website
Instalar
Aprenda más
Inventario
stock
Instalado
Aprenda más
Contabilidad
account_accountant
Actualizar
Aprenda más
Compra
purchase
Instalar
Aprenda más

libroSimple MFH

Por Rafael Reina

Instalar

Información | Datos técnicos

Sitio web	http://www.ynext.cl	**Nombre técnico**	libroSimple
Categoría	account.payment	**Licencia**	Affero GPL-3
Resumen	Libro Simple.	**Última versión**	15.0.13.0.0.1.0

PROCESO DE NEGOCIO

Un cliente desea realizar una compra de libros. En un primer contacto el cliente llama por teléfono a la librería para recibir información sobre los libros disponibles. El responsable de la librería atiende la llamada y registra los datos de contacto del posible cliente. Una vez satisfecha las necesidades de información del cliente, se analizan las necesidades del mismo.

Si el cliente muestra interés por las soluciones proporcionadas por la librería, se elabora un presupuesto para estimar el coste. Una vez aceptado el presupuesto por parte del cliente se realiza el cobro del mismo.

Historia de Usuario: gestión de ventas

COMO **Responsable de la librería**

QUIERO **automatizar la atención al cliente sobre libros**

PARA **enviar presupuesto por email y que se genere la factura una vez se haya confirmado por parte de este**.

RF 01: Registrar cliente.
RF 02: Elaborar presupuesto.
RF 03: Realizar cobro de factura.

RI 01: Datos del cliente.
RI 02: Presupuesto.
RI 03: Factura

Historia de Usuario: gestión de libros

COMO **Responsable de la librería**

QUIERO **automatizar la gestión de libros**

PARA **enviar presupuesto por email y que se genere la factura una vez se haya confirmado la compra de libros**.

RF 04: Registrar libro.
RF 05: Actualizar libro.
RF 06: Eliminar libro.

RI 04: Libro

Odoo

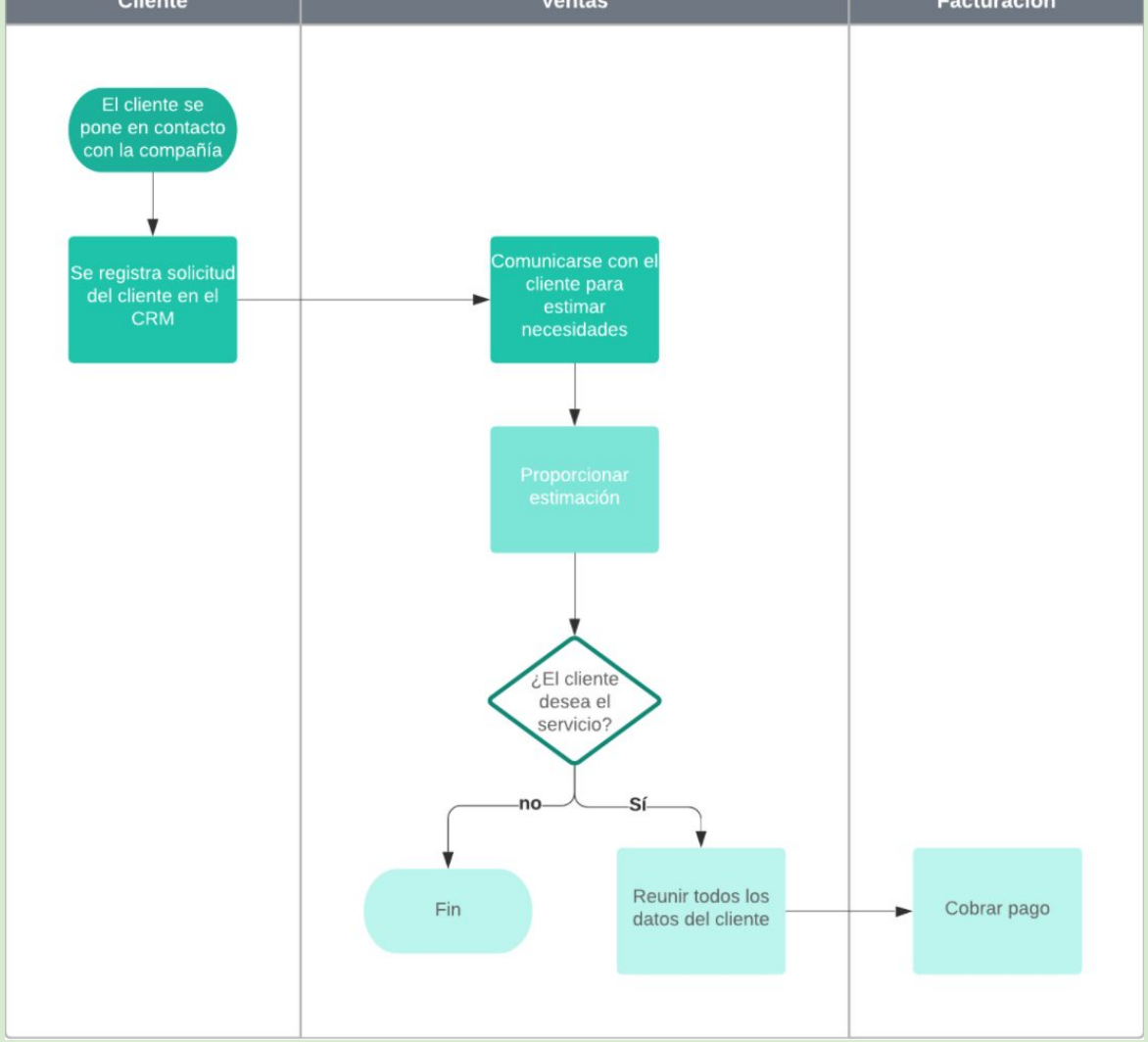
Módulo CRM

Módulo de Ventas

(incluido Factura y Productos)

Adaptación: **añadir el precio de venta en la vista del producto.**

Diseño: módulo de gestión de libros.



__init__.py del addons

Se encarga de cargar en el sistema las definiciones de los objetos del módulo.

Generalmente sólo es necesaria la carga del fichero que define el módulo principal.

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

from . import models
```

__manifest__.py: propiedades para el sistema

```
# -*- coding: utf-8 -*-
{
    'name': "miModuloLibros",
    'description': "ejemplo de desarrollo módulo simple para SGE 2021",
    'author': "Rafael Reina",
    'version': '1.0',
    'depends': ["base"],
    'demo': [],
    'test': [],
    'category': "Uncategorized",
    'data': ['views/milibro_view.xml', 'security/ir.model.access.csv'],
    'installable': True,
    'auto_install': False,
}
```

El modelo

Es una clase en Python que se ubica en el directorio models y que define los datos de los objetos del modelo de la aplicación.

ORM creará a partir de dicha definición la tabla en la base de datos para almacenar los datos de los objetos.

El modelo: milibro.py

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

from odoo import models, fields, api
class milibro(models.Model):
    _name = 'milibro'
    titulo = fields.Text(string='titulo', required=True)
    autor = fields.Text(string='autor', required=True)
    editorial = fields.Text(string='editorial', required=True)
    paginas = fields.Integer(string='paginas', required=True)
```

models/__init__.py

```
# -*- coding: utf-8 -*-
from . import milibro
```

Acciones: milibro_view.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<odoo>
<!-- Comentario en la Views -->
    <record id="view_ej_milibro_form" model="ir.ui.view">
      <field name="name">ej.milibro.form</field>
      <field name="model">ej.milibro</field>
      <field name="arch" type="xml">
        <form string="Listado de Milibro">   <group>
              <field name="titulo"/>
              <field name="autor"/>
              <field name="editorial"/>
              <field name="paginas"/>
          </group>          </form>       </field>  </record>
```

Vista Tree: milibro_view.xml

```
<record id="view_tree_milibro" model="ir.ui.view">
  <field name="name">ej.milibro.tree</field>
  <field name="model">ej.milibro</field>
  <field name="arch" type="xml">
    <tree>
        <field name="titulo"/>
        <field name="autor"/>
        <field name="editorial"/>
        <field name="paginas"/>
    </tree>
  </field>
</record>
```

Vista Tree: milibro_view.xml

```
    <record model="ir.actions.act_window" id="act_ej_milibro">
        <field name="name">milibro</field>
        <field name="res_model">ej.milibro</field>
        <field name="view_mode">tree,form</field>
        <field name="view_id"  ref="view_tree_milibro" />
    </record>
<!--  Declaramos los menu -->
<menuitem id="ej_milibro_menu" name="Milibro"
web_icon="stock,static/description/icon.png" sequence="10"/>
<menuitem id="submenu_ej_milibro_menu" name="Milibro" sequence="10"
parent="ej_milibro_menu"/>
<menuitem id="submenu_ej_milibro_action" name="Milibro" sequence="10"
parent="submenu_ej_milibro_menu" action="act_ej_milibro"/>
</odoo>
```

Vista form: milibro_view.xml

```
<record model="ir.ui.view" id="milibro_form_view">
      <field name="name">milibro.form</field>
      <field name="model">milibro</field>
      <field name="type">form</field>
      <field name="arch" type="xml">
            <tree string="Libros">
                  <field name="titulo"/>
                  <field name="paginas"/>
                  <field name="autor"/>
                  <field name="editorial"/>
            </tree>
      </field>
   </record>
```

Extender el modelo anterior: milibro2.py

Añadiremos más datos al objeto básico “milibro” y un nuevo tipos de objetos “categoria” que permita modelar que una categoría tiene muchos libros asociados.

- Clase “categoria”
- Clase “libro2” que hereda de libro
- Relación de “libro2” con “categoria” de “muchos objetos de libro a uno de categoria” con categoria

milibro2.py clase milibro2_categorias

```
# -*- coding: utf-8 -*-
from odoo import models, fields, api
class milibro2_categorias(models.Model):
        _name='milibro2.categorias'
        'name': fields.char('Descripcion', size=150, requiered=True),
```

milibro2.py clase milibro2 (1/)

```
class milibro2(models.Model):
    _name = 'milibro2'
    _inherit='milibro'
      'isbn': fields.char('ISBN', size=15),
      'precio':fields.float('PVP', digits=(4,2)),
      'resumen': fields.text('Descripcion'),
      'fecha': fields.date('Fecha'),
      'revisado': fields.boolean('Revisado'),
      'aprobado': fields.selection(('S','Si'), ('N','No'), ('P','Pendiente'), 'Aprobado'),
      'categoria': fields.many2one('milibro2.categorias','Categoria',
ondelete='cascade')
```

milibro2_view.xml (1/7)

```
<odoo> <data><!-- actions opening views on models -->
    <record model="ir.actions.act_window" id="action_milibro2_ cat_form">
      <field name="name">milibro2_cat</field>
      <field name="res_model">milibro2.categorias</field>
    </record>

    <menuitem name="Libros2" icon="terp-project" web_icon="data/libro.png"
web_icon_hover="data/libro.png" id="milibro2_menu"/>
    <menuitem name="Categorias" icon="terp-project" id="milibro2_cat_form"
parent="milibro2_menu" action="action_milibro2_cat_form"/>
```

milibro2_view.xml (2/ 7)

```
<record model="ir.ui.view" id="milibro2_cat_tree_view">
  <field name="name">milibro2.categorias.tree</field>
  <field name="model">milibro2.categorias</field>
  <field name="type">tree</field>
  <field name="arch" type="xml">
        <tree string="Categorias">
              <field name="name"/>
        </tree>
  </field>
</record>
```

milibro2_view.xml (3/ 7)

```
<record model="ir.ui.view" id="milibro2_cat_form_view">
  <field name="name">milibro2.categoria.form</field>
  <field name="model">milibro2.categorias</field>
  <field name="type">form</field>
  <field name="arch" type="xml">
        <tree string="Categorias">
              <field name="name"/>
        </tree>
  </field>
</record>
```

milibro2_view.xml (4/ 7)

```
<record model="ir.ui.view" id="milibro2_cat_form_view">
  <field name="name">milibro2.categoria.form</field>
  <field name="model">milibro2.categorias</field>
  <field name="type">form</field>
  <field name="arch" type="xml">
        <tree string="Categorias">
              <field name="name"/>
        </tree>
  </field>
</record>
```

milibro2_view.xml (5/7)

```
    <record model="ir.actions.act_window" id="action_milibro2_form">
      <field name="name">milibro2</field>
      <field name="res_model">milibro2</field>
    </record>

    <menuitem name="Mis Libros 2" icon="terp-project" id="milibro2_form"
parent="milibro2_menu" action="action_milibro2_form"/>
```

milibro2_view.xml (6/ 7)

```
<record model="ir.ui.view" id="milibro2_cat_tree_view">
  <field name="name">milibro2.tree</field>
  <field name="model">milibro2</field>
  <field name="type">tree</field>
  <field name="arch" type="xml">
        <tree string="Libros">
              <field name="titulo"/>
              <field name="autor"/>
        </tree>
  </field>
</record>
```

milibro2_view.xml (7/ 7)

```
<record model="ir.ui.view" id="milibro2_form_view">
    <field name="name">milibro2.form</field>
    <field name="model">milibro2</field>
    <field name="type">form</field>
    <field name="arch" type="xml">  <tree string="Libros">
                <field name="titulo"/> <field name="paginas"/>
                <field name="autor"/> <field name="isbn"/>
                <field name="PVP"/> <field name="fecha"/>
                <field name="revisado"/> <field name="aprobado"/>
                <field name="categoria"/> <field name="resumen"/>
            </tree>
    </field>
  </record> </data> </odoo>
```

Seguridad de acceso al modelo: **security/ir.model.access.csv**

Para poder usar el módulo una vez instalado es necesario poder acceder al modelo del mismo.

```
id,name,model_id:id,group_id:id,perm_read,perm_write,perm_create,perm_unlink

access_milibro,milibro,model_milibro,base.group_user,1,1,1,1

access_milibro2_categorias,milibro2.categorias,model_milibro2_categorias,base.group_user,1,1,1,1
```

Modificamos __manifest__.py

```
# -*- coding: utf-8 -*-
{
    'name': "Mi Modulo de Libros BASE",
    'description': "ejemplo de desarrollo módulo simple para SGE 2021",
    'author': "Rafael Reina",
    'version': '1.0',
    'depends': ["base", "milibro"],
    'init_xml':[],
    'update_xml':"milibro2_view.xml",
    'category': "Uncategorized",
    'active': True,
    'installable':True
        'data': [
         'security/ir.model.access.csv',
    ],
}
```

Instalar módulo desde el Terminal

/opt/odoo/odoo/odoo-bin -i milibro -c /opt/odoo/odoo/debian/odoo.conf -d 2021_milibro

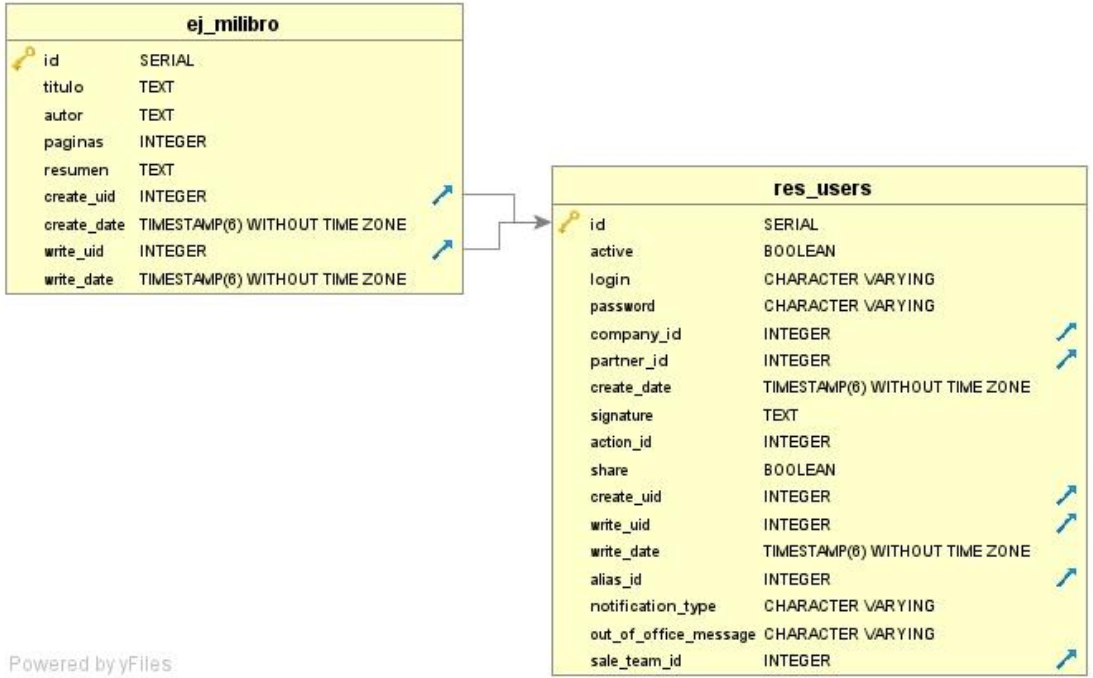
-i milibro es el módulo

-d 2021_milibro es la base de datos

Creación Tablas (ORM)

Se crean automáticamente a partir del modelo Python.

Script SQL en notas orador.



SQL Creación Tablas

```
CREATE TABLE ej_milibro (id SERIAL NOT NULL, titulo TEXT NOT NULL, autor TEXT NOT NULL, paginas INTEGER NOT NULL, resumen TEXT NOT NULL, create_uid INTEGER, create_date TIMESTAMP(6) WITHOUT TIME ZONE, write_uid INTEGER, write_date TIMESTAMP(6) WITHOUT TIME ZONE, PRIMARY KEY (id), CONSTRAINT ej_milibro_write_uid_fkey FOREIGN KEY (write_uid) REFERENCES "res_users" ("id") ON DELETE SET NULL, CONSTRAINT ej_milibro_create_uid_fkey FOREIGN KEY (create_uid) REFERENCES "res_users" ("id") ON DELETE SET NULL);
COMMENT ON TABLE ej_milibro IS 'ej.milibro';
COMMENT ON COLUMN ej_milibro.titulo IS 'titulo';
COMMENT ON COLUMN ej_milibro.autor IS 'autor';
COMMENT ON COLUMN ej_milibro.paginas IS 'paginas';
COMMENT ON COLUMN ej_milibro.resumen IS 'resumen';
COMMENT ON COLUMN ej_milibro.create_uid IS 'Created by';
COMMENT ON COLUMN ej_milibro.create_date IS 'Created on';
COMMENT ON COLUMN ej_milibro.write_uid IS 'Last Updated by';
COMMENT ON COLUMN ej_milibro.write_date IS 'Last Updated on';
```

Odoo 14

Instalar el nuevo módulo

1. Comprobar las rutas de addons de los ficheros de configuración de odoo que usa el servidor (/etc/odoo/odoo.conf en mi caso)
2. Añadir la ruta del addons donde se van a añadir los nuevos addons (/opt/odoo/odoo/addons en mi caso).
3. Reiniciar el servidor.
4. Actualizar la lista de aplicaciones.

TAREA DESARROLLO DEL PROYECTO

Documento de partida: Historias de Usuario y Casos de Uso (REQUISITOS)

Cada miembro del equipo elegirá una de las historias de usuario y un módulo específico para Odoo. Se deben describir el código generado para:

- MODELO y BASE DE DATOS
- VISTAS
- ACCIONES DE LAS VISTAS

Despliegue de Odoo

https://www.odoo.com/documentation/14.0/setup/deploy.html#

Práctica

https://www.python.org/

https://docs.python.org/3/

Tutorial de Python 3

https://docs.python.org/es/3/tutorial/index.html

https://tutorial.recursospython.com/

https://recursospython.com/guias-y-manuales/aplicacion-blockchain-desde-cero/

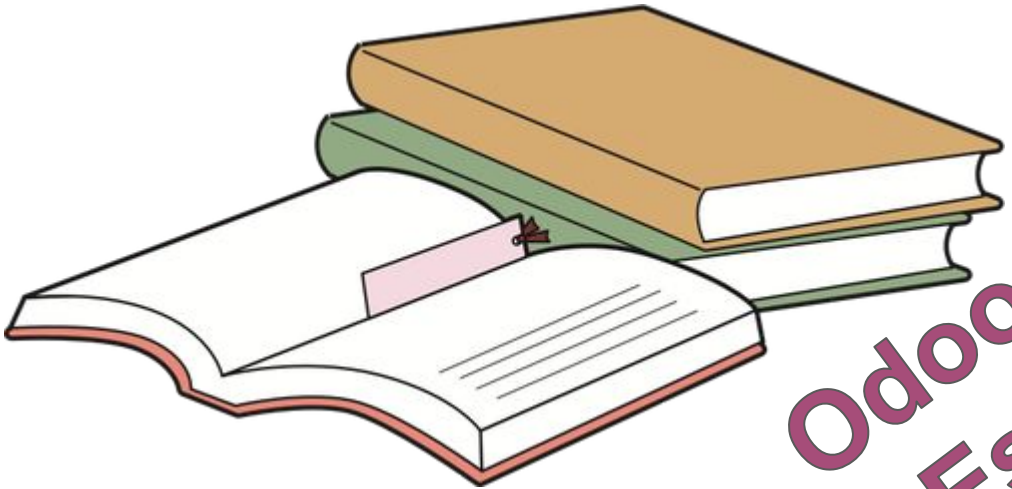
https://web.archive.org/web/20150209025803/http://csrc.nist.gov/groups/STM/cavp/documents/shs/sha256-384-512.pdf

https://demoblockchain.org/hash

https://medium.com/instanz/qu%C3%A9-es-ethereum-a0e39404200e (ver docker.hub)

Webgrafía

https://fundamentos-de-desarrollo-en-odoo.readthedocs.io/es/latest/odoo-development-essentials.html



Odoo Development Essentials Book