

UT6. Desarrollo de componentes

Sistemas de Gestión Empresarial

2º CFGS DAM - Curso 2021/2022

ÍNDICE

1.- DESARROLLANDO UN MÓDULO EN ODOO

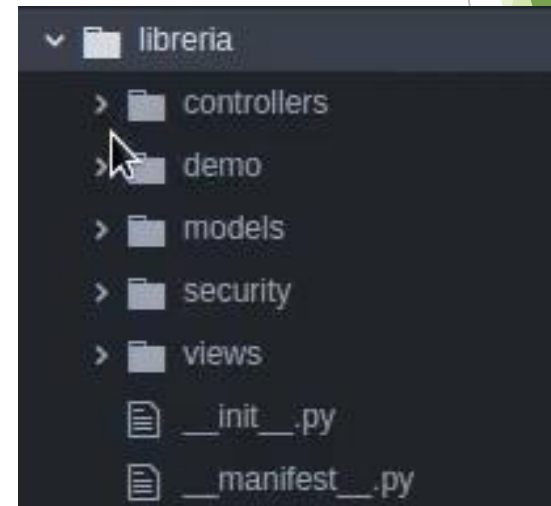
- 1.1. Estructura de un módulo
- 1.2. Crear modelos
- 1.3. Creación de las vistas
- 1.4. Creación de las relaciones
- 1.5. Vistas de búsquedas
- 1.6. Campos calculados
- 1.7. Creación de informes
- 1.8. Permisos
- 1.9. Retoques finales: iconos y datos precargados

1.1. ESTRUCTURA DE UN MÓDULO

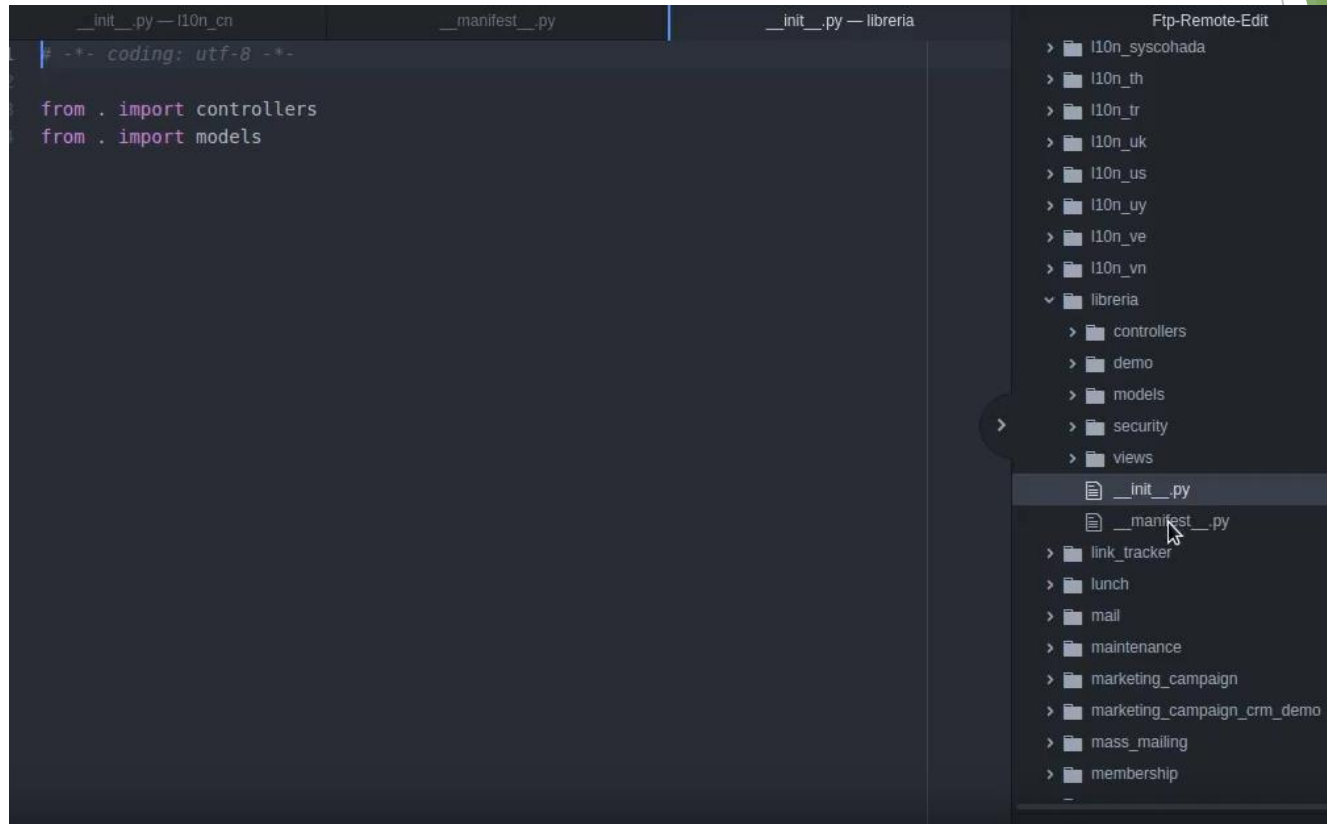
- Los módulos se van a crear dentro de la carpeta **addons**, que se encuentra dentro del directorio odoo (/opt/odoo/addons).
- Desde el directorio odoo ejecutamos el siguiente comando, que creará dentro del directorio addons la estructura del módulo.

Con el usuario
odoo

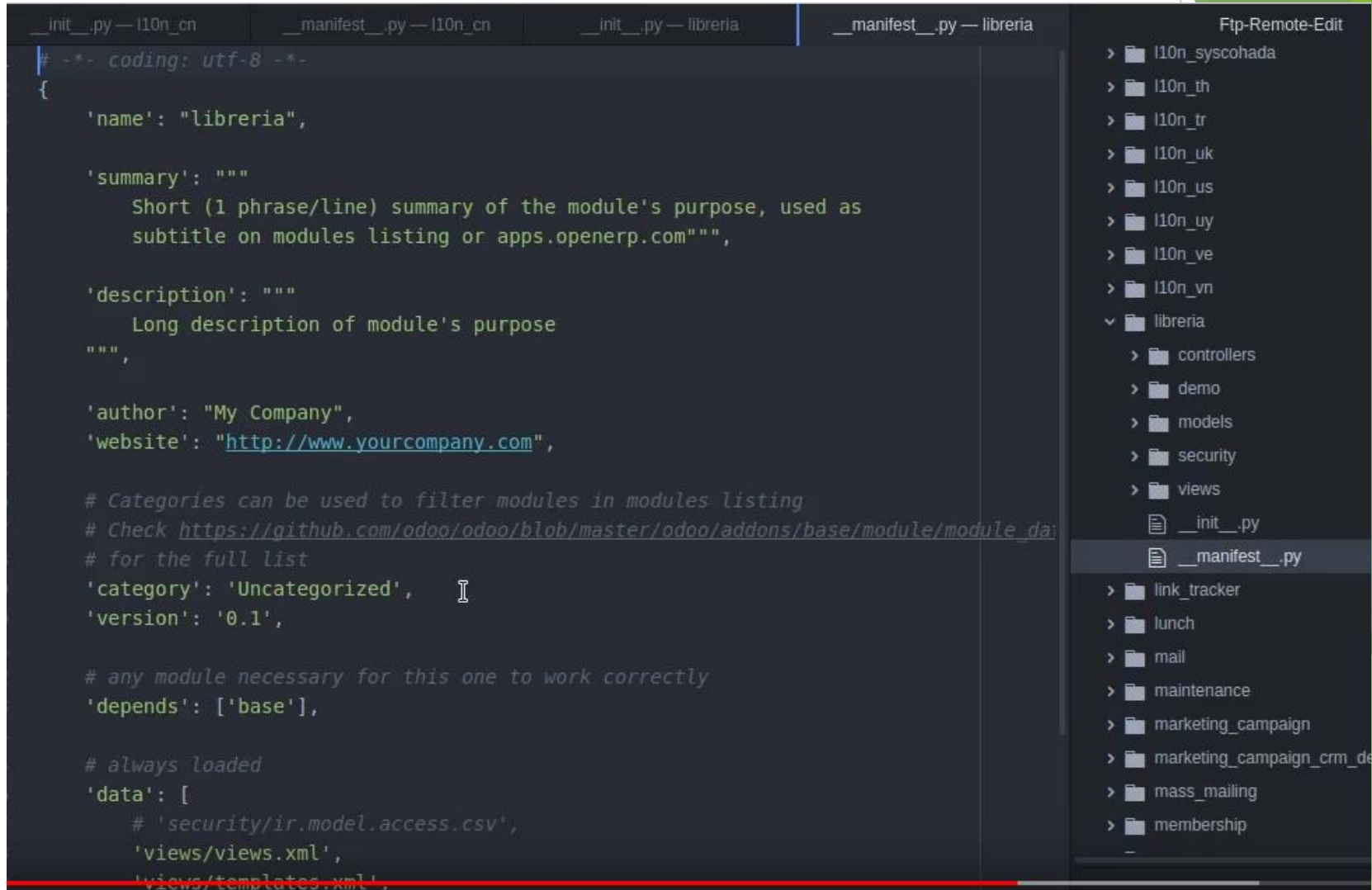
```
./odoo-bin scaffold libreria addons
```



1.1. ESTRUCTURA DE UN MÓDULO



1.1. ESTRUCTURA DE UN MÓDULO



The image shows a screenshot of an FTP client interface. The main window displays the file structure of a module named 'libreria'. The left pane shows the file list, and the right pane shows the contents of the selected file, which is the 'manifest.py' file. The 'manifest.py' file contains the following code:

```
__init__.py — l10n_cn    __manifest__.py — l10n_cn    __init__.py — libreria    __manifest__.py — libreria
# -*- coding: utf-8 -*-
{
    'name': "libreria",

    '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': "http://www.yourcompany.com",

    # Categories can be used to filter modules in modules listing
    # Check https://github.com/odoo/odoo/blob/master/odoo/addons/base/module/module_data.py
    # for the full list
    'category': 'Uncategorized',
    'version': '0.1',

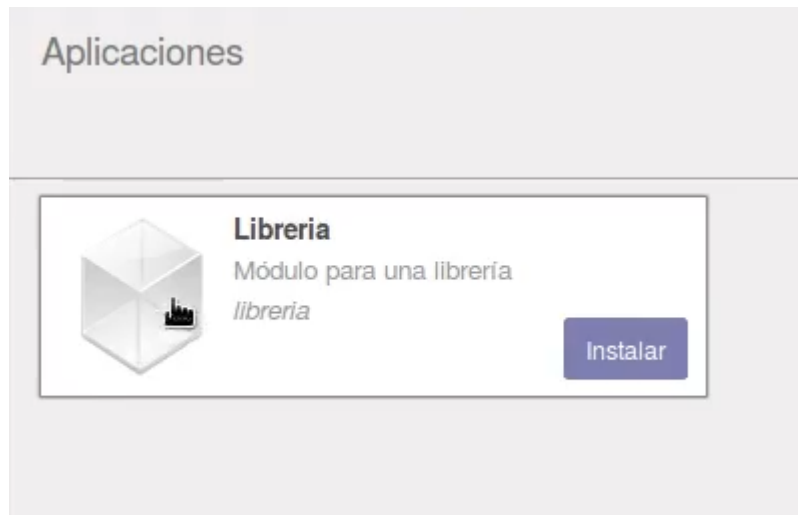
    # any module necessary for this one to work correctly
    'depends': ['base'],

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

The right pane shows the file list for the 'libreria' module, including folders like 'controllers', 'demo', 'models', 'security', 'views', and files like '__init__.py' and '__manifest__.py'.

1.1. ESTRUCTURA DE UN MÓDULO

ACTUALIZAR LA LISTA DE LAS APLICACIONES



1.2. CREAR MODELOS

- - En la carpeta **models**, dentro del archivo **models.py**.

```
from odoo import models, fields, api

class libreria_categoria(models.Model):
    _name = "libreria_categoria"

    name = fields.Char(string="Nombre", required=True, help="Introduce el nombre de la categoría")
    descripcion = fields.Text(string="Descripción")

class libreria_libro(models.Model):
    _name = "libreria_libro"

    name = fields.Char(string="Título", required=True)
    precio = fields.Float(string="Precio")
    ejemplares = fields.Integer(string="Ejemplares")
    fecha = fields.Date(string="Fecha de compra")
    segmano = fields.Boolean(string="Segunda mano")
    estado = fields.Selection([( '0', 'Bueno'), ( '1', 'Regular'), ( '2', 'Malo')], string="Estado", default="0")
```

Reiniciar el servidor, cada vez que
cambiamos un archivo .py.
systemctl restart odoo.service

1.3. CREACIÓN DE LAS VISTAS

- - En la carpeta **views**, dentro del archivo **views.xml**
- **VISTA TIPO ÁRBOL**

```
<odoo>
  <data>
    <record model="ir.ui.view" id="libreria.categoria_tree">
      <field name="name">libreria.categoria.tree</field>
      <field name="model">libreria.categoria</field>
      <field name="arch" type="xml">
        <tree>
          <field name="name"/>
          <field name="descripcion"/>
        </tree>
      </field>
    </record>
```

El modelo tiene que ser el mismo que el `_name` definido anteriormente

1.3. CREACIÓN DE LAS VISTAS

- - En la carpeta **views**, dentro del archivo **views.xml**
- **VISTA TIPO FORMULARIO**

```
<record model="ir.ui.view" id="libreria.categoria_form">
  <field name="name">libreria.categoria.form</field>
  <field name="model">libreria.categoria</field>
  <field name="arch" type="xml">
    <form>
      <group colspan="2" col="2">
        <field name="name"/>
        <field name="descripcion"/>
      </group>
    </form>
  </field>
</record>
```

1.3. CREACIÓN DE LAS VISTAS

- - En la carpeta **views**, dentro del archivo **views.xml**
- **ACCIÓN DE VENTANA**

```
<record model="ir.actions.act_window" id="libreria.categoria_action_window">  
  <field name="name">libreria.categoria.action_window</field>  
  <field name="res_model">libreria.categoria</field>  
  <field name="view_mode">tree,form</field>  
</record>
```

MENÚ

```
<menuitem name="Libreria" id="libreria.menu_root"/>  
  
<menuitem name="Categorías" id="libreria.categoria_menu" parent="libreria.menu_root"  
action="libreria.categoria_action_window"/>
```

1.4. CREACIÓN DE LAS RELACIONES

- - 1 categoría - muchos libros
- 1 libro - 1 categoría

```
class libreria_libro(models.Model):  
    _name = "libreria.libro"  
  
    name = fields.Char(string="Título", required=True)  
    precio = fields.Float(string="Precio")  
    ejemplares = fields.Integer(string="Ejemplares")  
    fecha = fields.Date(string="Fecha de compra")  
    segmano = fields.Boolean(string="Segunda mano")  
    estado = fields.Selection([('0', 'Bueno'), ('1', 'Regular'), ('2', 'Malo')], string="Estado", default="0")  
    categoria = fields.Many2one("libreria.categoria", string="Categoría", required=True, ondelete="cascade")
```

```
class libreria_categoria(models.Model):  
    _name = "libreria.categoria"  
  
    name = fields.Char(string="Nombre", required=True, help="Introduce el nombre de la categoría")  
    descripcion = fields.Text(string="Descripción")  
    libros = fields.One2many("libreria.libro", "categoria", string="Libros")
```

1.4. CREACIÓN DE LAS RELACIONES

- - Modificamos ahora views.xml
- - Añadiendo al libro la categoría:

```
<field name="categoria"/>
```

- Añadiendo a categoría los libros:

```
<field name="libros">  
  <tree>  
    <field name="name"/>  
    <field name="precio"/>  
    <field name="ejemplares"/>  
  </tree>  
</field>
```

1.5. VISTAS DE BÚSQUEDA

```
<record model="ir.ui.view" id="libreria.libro_search_view">
  <field name="name">libreria.libro.search</field>
  <field name="model">libreria.libro</field>
  <field name="arch" type="xml">
    <search>
      <field name="name" string="Título"/>
      <field name="categoria" string="Categoría"/>
      <filter name="baratos" domain="(['precio','<=',5)"]"/>
    </search>
  </field>
</record>
```

Dentro de la etiqueta search, podemos filtrar por un campo, por ejemplo, el título, la categoría... O crear filtros, se le establece un nombre y un dominio, que consta de tres partes: campo al que se le va a aplicar el filtro, comparación y valor.

1.5. VISTAS DE BÚSQUEDA

Operador	Descripción
>	Más grande que
<	Más pequeño que
>=	Más grande o igual que
<=	Más pequeño o igual que
=	Igual que
!=	Diferente de
like	Comparador de cadenas que distingue entre mayúsculas y minúsculas
ilike	Comparador de cadenas que no distingue entre mayúsculas y minúsculas
in	Contenido en (comparar un valor con una lista de valores)
not in	No contenido en (comparar un valor con una lista de valores)
not	Negación (operador booleano que invierte el valor)
child_of	Comparador utilizado en estructuras de tipo árbol

Ejemplos:

- `[('qty_available', '<', 10)]` – utilizado en la lista de productos para mostrar solo los productos con cantidad en stock menor que 10
- `[('state', 'ilike', 'confirmed')]` – utilizado en la lista de ventas para mostrar solo las ventas confirmadas
- `[('categ_id', 'child_of', 5)]` – utilizado en la lista de categorías de producto para mostrar solo las categorías que tienen como padre la categoría 5

1.6. CAMPOS CALCULADOS

Añadir un campo calculado al modelo biblioteca, que sea el importe total: cantidad * precio.

- Añadir un método dentro de libros en models.py que haga el cálculo.

```
importetotal = fields.Float(string="Importe total", compute="_importetotal", store=True)
```

- **compute:** indicamos la función privada que va a realizar el cálculo.
- **store:** indicamos que se calcule y se guarde. Sólo se vuelve a calcular, si se modifican los datos.
- Implementamos la función:

1.6. CAMPOS CALCULADOS

- Implementamos la función:

```
@api.depends('precio','ejemplares')
def _importetotal(self):
    for r in self:
        r.importetotal = r.ejemplares*r.precio
```

@api.depends - Es una anotación de odoo, que le está indicando que sólo llame a la función si cambia los valores de precio o de ejemplares.

En la función siempre poner el **bucle for**, para que funcione si se le pasa una lista.

- Sólo faltaría meter ese campo en la **vista** formulario

```
<field name="importetotal"/>
```


1.7. CREACIÓN DE INFORMES

1. Crear dentro del módulo un directorio reports donde se van a guardar los informes
2. Crear el archivo report_libro.xml donde se va a definir.

```
/libreria/reports/report_libro.xml
```

1. El archivo consta de dos partes:
Campo report - Se va a definir el informe

```
<data>
  <report
    id="libreria.report_libro"
    model="libreria.libro"
    string="Informe libro"
    name="libreria.report_libro_view"
    file="libreria.report_libro_view"
    report_type="qweb-pdf"/>
```

1.7. CREACIÓN DE INFORMES

```
<template id="report_libro_view">
  <t t-call="web.html_container">
    <t t-foreach="docs" t-as="libro">
      <t t-call="web.external_layout">
        <div class="page">
          <h2 t-field="libro.name"/>
          <div>
            <strong>Precio:</strong>
            <span t-field="libro.precio"/>
          </div>
          <div>
            <strong>Ejemplares:</strong>
            <span t-field="libro.ejemplares"/>
          </div>
          <div>
            <strong>Categoría:</strong>
            <span t-field="libro.categoria"/>
          </div>
        </div>
      </t>
    </t>
  </t>
</template>
</data>
</odoo>
```

Campo template - Se va a definir el contenido del informe

1.7. CREACIÓN DE INFORMES

Modificar `__manifest__.py`, para que actualice también el informe cuando le demos a actualizar el módulo.

```
# always loaded
'data': [
    # 'security/ir.model.access.csv',
    'views/views.xml',
    'views/templates.xml',
    'reports/report_libro.xml'
],
```

1.8. PERMISOS

1. En el directorio security crear un archivo security.xml, donde añadiremos un `<record>` por cada rol que vayamos a tener.

```
<odoo>
  <data>
    <record id="libreria_manager" model="res.groups">
      <field name="name">Librería / Responsable</field>
    </record>
  </data>
</odoo>
```

1.8. PERMISOS

2. En el fichero `ir.model.access.csv` se define los permisos que va a tener.

```
id,name,model_id:id,group_id:id,perm_read,perm_write,perm_create,perm_unlink
categoria_responsable,categoria responsable,model libreria categoria,libreria manager,1,1,1,1
libro_responsable,libro responsable,model libreria libro,libreria manager,1,1,1,1
```

La primera línea es una cabecera así que no se modifica.

Resto de líneas:

id_regla - sin espacios

nombre_regla - puede contener espacios

model_id - siempre empieza por `model_` seguido de la tabla sobre la que le vamos a dar permisos

group id - a qué grupo le estamos dando permisos

perm read, perm write, perm create y perm unlink - 0 o 1

1.8. PERMISOS

3. Añadir security.xml en el archivo __manifest__.py para que lo cargue

```
# always loaded
'data': [
    'security/security.xml',
    'security/ir.model.access.csv',
    'views/views.xml',
    'views/templates.xml',
    'reports/report_libro.xml'
],
```

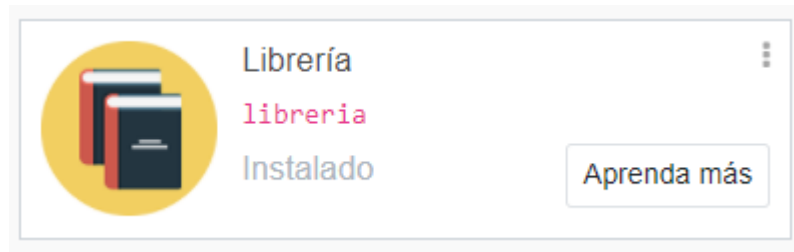
1.8. PERMISOS

4. Añadir en el menú padre y en los hijos que se encuentran en el archivo views.xml, que sólo se muestren para el grupo libreria_manager.

```
<menuitem name="Libreria" id="libreria.menu_root" groups="libreria_manager"/>
```

1.9. RETOQUES FINALES

- AÑADIR UN ICONO
1. Crear dentro del directorio libreria, un directorio llamado **static**, y dentro de static, un directorio llamado **description**.
 2. Guardar la imagen **icon.png** en el directorio description



1.9. RETOQUES FINALES

- PRECARGA DE DATOS

1. Crear directorio data y dentro de él el fichero data.xml
2. Añadirlo al fichero __manifest__.py para que lo cargue.

```
# always loaded
'data': [
    'security/ir.model.access.csv',
    'security/security.xml',
    'views/views.xml',
    'views/templates.xml',
    'reports/report_libro.xml',
    'data/data.xml'
],
```

1.9. RETOQUES FINALES

- PRECARGA DE DATOS

3. Escribir el fichero data.xml, añadiendo un record por cada campo a cargar, indicando un identificador, la tabla, el campo y el valor.

```
<odoo>
  <data>
    <record id="cat1" model="libreria.categoria">
      <field name="name">Novela</field>
    </record>

    <record id="cat2" model="libreria.categoria">
      <field name="name">Ensayo</field>
    </record>
  </data>
</odoo>
```

4. Reiniciamos odoo
Desinstalamos la aplicación
Actualizamos la lista de aplicaciones y volvemos a instalarla

ANEXOS

Cli Odoo-bin

- ▶ Otra forma de actualizar un módulo (ejecutarlo con un usuario existente en la bd, en nuestro caso odoo)

Control + c parar odoo

`./odoo-bin -d nombrebd -u nombremodulo`

Refrescar la página

- ▶ Para que se detecten de forma automática los cambios (actualiza la app y reinicia el servidor)

*si está arrancado previamente, hay que pararlo (`systemctl stop odoo`)

`su odoo`

`./odoo-bin --dev all`

Refrescar la página

Restricciones

- ▶ Las restricciones la añadimos en las clases Python
- ▶ Existen dos tipos
 - ▶ **Restricciones SQL**

Por ejemplo no queremos que se puede repetir el nombre de las categorías

```
_sql_constraints = [('name_unique', 'unique(name)', 'La categoría ya existe')]
```

- ▶ **Restricciones Python**

```
from odoo.exceptions import ValidationError
```

```
...
```

```
@api.constrains('date_end')
```

```
def _check_date_end(self):
```

```
    for record in self:
```

```
        if record.date_end < fields.Date.today():
```

```
            raise ValidationError("La fecha de fin debe ser posterior a hoy")
```