

## Unidad 7

---



# Desarrollo de componentes

## Sistemas de gestión empresarial



# Índice



- 7.1. Estructura básica de un módulo de Odoo. El comando scaffold**
- 7.2. Generación de un módulo**
  - 7.2.1. Creación de un módulo en el comando scaffold
  - 7.2.2. Creación del modelo
  - 7.2.3. Diseño de las vistas básicas
  - 7.2.4. Acciones de ventana
  - 7.2.5. Opciones de menú
- 7.3. Generación de informes**
- 7.4. Permisos del módulo**
- 7.5. Creación paso a paso de un módulo**



## Introducción

En este capítulo se trabajará con ficheros .py (Python), .xml y .csv directamente. Para ello, se recomienda utilizar un software de procesador de textos adecuado ya que el código se modificará, como Sublime Text, Visual Studio Code o Atom o, incluso de un ide como Pycharm.

El entorno utilizado en este texto es el de un servidor Odoo instalado en una máquina virtual con sistema operativo Ubuntu server. Las conexiones entre la máquina virtual Ubuntu Server donde corre Odoo y el host (una máquina Windows 10) se han realizado mediante navegador y con PuTTY para conexiones ssh. Para explicar los procedimientos y obtener las imágenes, se ha utilizado Atom con un plugin de acceso remoto, que ha permitido trabajar directamente en el sistema de ficheros del servidor.

Dependiendo de las versiones de los diferentes paquetes de software utilizados pueden existir ligeros cambios en el proceso.

## Al finalizar esta unidad

- + Aprenderemos a implantar la programación de módulos y elaborar informes.
- + Conoceremos la estructura que tiene un Módulo o Aplicación de Odoo.
- + Seremos capaces de usar el comando scaffold.
- + Estudiaremos los diferentes modelos de una aplicación y los diversos tipos de vista.
- + Realizaremos la codificación básica en Python.



# 7.1.

## Estructura básica de un módulo de Odoo. El comando scaffold

En las unidades anteriores hemos visto que es posible realizar desarrollos y modificaciones utilizando la parte gráfica del sistema ERP.

En este tema, veremos el desarrollo de módulos en Odoo desde cero. Cada aplicación y módulo de Odoo se almacena en su propia carpeta, dentro de la carpeta de addons.

El nombre técnico del módulo corresponde al nombre del directorio donde está almacenado en el directorio addons o en cualquier especialmente creado para ello.

En general, un módulo de Odoo consta de al menos dos archivos principales y una serie mínima de directorios que siguen una estructura común:

Estructura de un módulo Odoo		
Ficheros	<code>__init__.py</code>	Archivo principal del módulo y de carga del código Python
	<code>__manifest__.py</code>	Archivo de manifiesto del módulo con metadatos y declaración de ficheros XML que deben ser cargados.
Directorios básicos	<code>models/</code>	Contiene archivos Python (.py) para definición de las entidades (modelos, clases y métodos). Se aconseja un archivo por modelo.
	<code>views/</code>	Contiene los archivos .xml que generan interfaces de usuario: vistas, acciones, elementos de menú, etc.
	<code>security/</code>	Contiene archivos .xml y csv que definen permisos de los grupos y las reglas de seguridad (listas de control de acceso). Los archivos que suelen estar en su interior son <code>ir.model.access.csv</code> y <code>security.xml</code> .
	<code>report/</code>	Contiene modelos de informes (.xml) y archivos .py y .xml relacionados
	<code>controllers/</code>	Contiene archivos de código para controladores y rutas http del sitio web
	<code>demo</code>	Utilizado para tener algunos datos de prueba precargados en ficheros .xml (su carga es opcional)
	<code>static</code>	Para documentación para el sitio web en directorios <code>img/</code> , <code>css/</code> , <code>js/</code> , <code>lib/</code> , etc. En este directorio se almacenan principalmente archivos JavaScript, hojas de estilo e imágenes. No se necesita mencionar en el manifiesto del módulo, pero se referencian en la plantilla web.
Otros directorios	<code>i18n/</code>	Almacena archivos referentes a la localización y traducción.
	<code>data/</code>	Archivos css y xml que cargan datos al sistema (en caso de tener contenido se carga obligatoriamente).
	<code>wizard/</code>	Almacena vistas de tipo wizard. Reagrupa modelos transitorios.
	<code>tests/</code>	Puede incluir códigos utilizados para realizar pruebas de funcionamiento.



### Archivo `_init_.py`

Archivo principal del módulo con instrucciones de importación: es el primer archivo que se lee durante el proceso de ejecución e indica qué otros archivos Python deben importarse y ejecutarse pues contiene las referencias a otros archivos `.py` y a los directorios donde se ubican.

Cualquier directorio que contenga archivos Python debe incluir este tipo de archivo, incluso si está vacío.

### Archivo `_manifest_.py`

Es otro archivo que contiene los datos básicos del módulo, como la descripción, información del autor, dependencias, así como los datos relacionados con la instalación, la seguridad y la existencia o no de datos demo. Es un diccionario Python clave-valor con metadatos del módulo.

El orden en que se declaran los archivos `.xml` es importante. Por ejemplo, si existe un archivo `.xml` para la vista y otro para las acciones de menú y ventana, es necesario que primero se declare el archivo de vistas y luego el archivo donde estén las acciones.

Los módulos cuyas carpetas se encuentran en la ruta de acceso de addons y se incluyen al crear una nueva base de datos se pueden instalar directamente y también desde la línea de comando.

Al agregar nuevos archivos a un módulo, recuerde declararlos en `_manifest_.py` (archivo de datos) o en `_init_.py` (archivo de código) para evitar que se ignoren y no se carguen.

## 7.2.

## Generación de un módulo

Mediante la creación manual de archivos y carpetas se puede crear un nuevo módulo completamente funcional. Sin embargo, Odoo proporciona un mecanismo para generar automáticamente esta estructura básica, incluyendo en algunos de los archivos líneas ejemplo del contenido que debe tener, lo cual es de gran ayuda para el programador.

### 7.2.1. Creación de un módulo en el comando `scaffold`

La creación de la estructura de un módulo base se puede hacer automáticamente usando el comando `scaffold` que, como apoyo, genera la cantidad mínima de archivos y carpetas necesarias para que el módulo sea reconocido por Odoo.

```
$ sudo odoo scaffold "nombre del módulo" "carpeta_
para_addons"/
```

Así, por ejemplo, para crear el módulo llamado `jig-modulo` en la misma carpeta donde están el resto de los módulos, con `odoo` se hace de la siguiente manera:

```
$ sudo odoo scaffold mi_modulo addons/
```

que genera el directorio `mi_modulo` con el contenido básico específico.

```
profesor@profesor-VirtualBox:~/odoo$ ./odoo-bin scaffold mi_modulo1 addons
profesor@profesor-VirtualBox:~/odoo$ ls
addons  debian  MANIFEST.in  README.md  setup
CONTRIBUTING.md  doc  odoo  requirements.txt  setup.cfg
COPYRIGHT  LICENSE  odoo-bin  SECURITY.md  setup.py
```

Imagen 1. Resultado del comando Scaffold

Como hemos estudiado, cada uno de los ficheros generados incluye un ejemplo del contenido que debe tener el fichero.

En `_init_.py` se incluyen las carpetas en las que debe buscar Python los archivos `.py` que se van a ejecutar.

```
__init__.py
1  # -*- coding: utf-8 -*-
2
3  from . import controllers
4  from . import models
5
6
```

Imagen 2. Contenido de `_init_.py`

En `_manifest_.py` se pueden incluir los datos básicos del módulo, que aparecerán en la caja Kanban del mismo.

```
__init__.py  __manifest__.py
1  # * coding: utf 8 *
2
3  {
4      'name': 'Módulo de la asignatura de SGE',
5      'summary': '''
6          Esto es un módulo de prueba''',
7      'description': '''
8          Módulo de prueba utilizado en la unidad 7 de la asignatura de Sistemas de Gestión Empresarial
9          ''',
10     'author': 'Universae',
11     'website': 'https://www.universae.com',
12
13     # Categories can be used to filter modules in modules listing
14     # Check https://github.com/odoo/odoo/blob/15.0/odoo/addons/base/data/ir_module_category_data.xml
15     # for the full list
16     'category': 'Uncategorized',
17     'version': '0.1',
18
19     # any module necessary for this one to work correctly
20     'depends': ['base'],
21
22     # always loaded
23     'data': [
24         # 'security/ir.model.access.csv',
25         'views/views.xml',
26         'views/templates.xml',
27     ],
28     # only loaded in demonstration mode
29     'demo': [
30         'demo/demo.xml',
31     ],
32 }
33
34
35
```

Imagen 3. Contenido de `_manifest_.py` ya modificado

Con este proceso, se actualiza la lista de Aplicaciones de Odoo, obteniendo la caja Kanban del módulo instalada con los datos introducidos en el `_manifest_.py` pero sin funcionalidad.





## 7.2.2. Creación del modelo

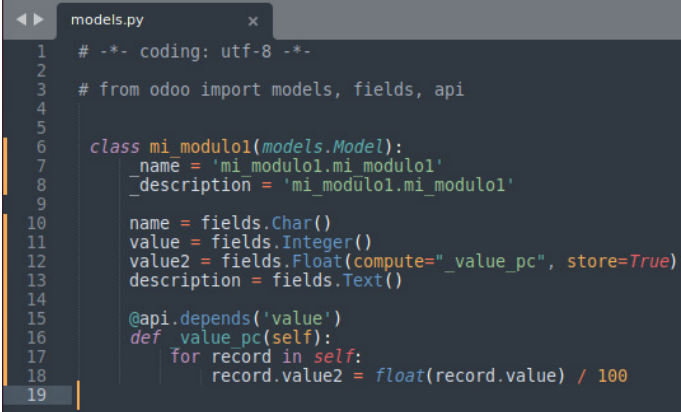
Para definir los modelos que emplea el módulo, se usa el archivo `models.py` de la carpeta `models`.

Cada objeto de Odoo corresponde a una clase Python. Siempre se extienden de la clase `Model` y deben nombrarlos incluyendo el nombre del módulo.

Los campos deben contener al menos:

- > **\_name:** las buenas prácticas recomiendan que siga la sintaxis: `mymodule.model`
- > **name:** de tipo `char` con atributo `string` con lo que se quiere instanciar. Este campo siempre debe existir y se utilizará en las vistas de búsqueda y para establecer relaciones entre objetos.

Además, se listarán el resto de los campos de configuración del modelo que se está generando con sus características.



```

1  # -*- coding: utf-8 -*-
2
3  # from odoo import models, fields, api
4
5
6  class mi_modulo1(models.Model):
7      _name = 'mi_modulo1.mi_modulo1'
8      _description = 'mi_modulo1.mi_modulo1'
9
10     name = fields.Char()
11     value = fields.Integer()
12     value2 = fields.Float(compute='_value_pc', store=True)
13     description = fields.Text()
14
15     @api.depends('value')
16     def _value_pc(self):
17         for record in self:
18             record.value2 = float(record.value) / 100
19

```

Imagen 4. Estado inicial de `models.py`

En caso de que existan varios modelos, es imprescindible establecer explícitamente las relaciones entre ellos, utilizando los campos `fields.Many2one`, `fields.Many2many` o `fields`.

En cuanto a los campos computados, se construyen usando un método dentro del objeto, el cual se encarga de realizar el cálculo con una propiedad de computación, que es la función privada que se usará para hacerlo. La forma de definirlo es:

`@api.depends ('campo1' , 'campo2' )`: Anotación (se ejecuta cuando cambien `campo1` o `campo2`).

`def _campocalculado(self)`: Se itera por si al método se le llama con varios registros.

`for r in self:`

`r.campocalculado = r.campo1 "operación" r.campo2`



### 7.2.3. Diseño de las vistas básicas

Se describen en el archivo `views.xml` de la carpeta `views`. En él se definen tanto las vistas como las opciones de menú y las acciones de ventana relacionadas.

Las vistas son objetos de tipo `record` de la tabla `ir.ui.view`. Su identificador (necesario) sigue la nomenclatura `nombredelmodulo.nombredelmodelo_tipo` donde `tipo` puede ser `form`, `tree`, `kanban`... (es importante que solo se utilice un único punto en el nombre).

Las líneas que debe incluir son:

```
<record model = "ir.ui.view" id= "nombredelmodulo.nombredelmodelo_tipo">
```

- > **Name:** `nombredelmodulo.nombredelmodelo_tipo` (su nombre en el atributo `id` cambiando el guion bajo por punto).
- > **model:** objeto sobre el que aplica.
- > **arch:** tipo de archivo.

Por ejemplo:

```
<record model="ir.ui.view" id="nombredelmodulo.nombredelobjeto_form">
```

```
    <field name = "name"> nombredelmodulo.nombredelobjeto_form</field>
```

```
    <field name = "model"> nombredelmodulo.nombredelobjeto</field>
```

```
    <field name="arch" type= "xml">
```

Según el tipo de vista, siguen la estructura siguiente:

#### Árbol

```
<tree>
<field name="name"/>
<field name="value"/>
```

...

```
< / tree>
```

#### Formulario

```
<form>
    <group colspan="x" col="y">

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

...

```
    < / group>
```

```
< / form>
```





### Búsqueda

Existe una vista de búsqueda por el campo name. Se pueden crear procesos de búsqueda por otros campos del modelo mediante la inclusión de código entre etiquetas

```
<search>

    <field name="name" string=""/>
    <field name="value" string=""/>
    ...
< / search>
```

Además, se puede añadir un filtro, es decir una condición sobre un campo.

Los filtros se construyen definiendo un dominio donde la condición está formada por campo, operador y valor.

## 7.2.4. Acciones de ventana

En este punto especificamos las acciones de ventana, con la siguiente estructura:

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

## 7.2.5. Opciones de menú

Los elementos de menú siguen una jerarquía de tipo árbol, y que especifican las acciones de ventana mediante el atributo "action=".

```
<menuitem name="nombredelmodulo"
id="nombredelmodulo.menu_root"/>

<menuitem name="nombremenu 1"
id="nombredelmodulo.nombremenu_1"
parent=" nombremodulo.menu_root"
action= "nombredelmodulo.action_window"/>

<menuitem name="nombremenu 2"
id="nombredelmodulo.nombremenu_2"
parent=" nombremodulo.menu_root"
action= "nombredelmodulo.action_window"/>
```



# 7.3.

## Generación de informes

Los informes en Odoo se generan en archivos xml utilizando Qweb. Para crear nuevos informes, debe guardarlos en una carpeta específica conocida como reports.

Una vez creados, se deben declarar en apartado 'data' del fichero \_manifest\_.py y agregar las dependencias necesarias en el mismo archivo, en el apartado 'depends'.

La estructura de un informe se compone de dos partes:

> `<record/>` donde se crea un registro de tipo `ir.action.report`:

```
<record id="nombre_informe" model="ir.action.report">
```

y se definen las características básicas generales del informe

```
<record id="nombre"model="ir.action.report">
```

» `model=` modelo sobre el que actúa

» `report_name=` modulo.template id de la parte `<template/>`

» `report_file=` modulo.template id de la parte `<template/>`

» `report_type="qweb-tipo"` donde tipo puede ser html o pdf

> `<template/>`, que especifica los campos que incluye y las plantillas utilizadas para generar el informe.

La vista de tipo Qweb que la contiene por defecto es `web.html_container`.

Además, el uso de `web.internal_layout` generará un pdf con encabezado y pie de página mínimos, mientras que al usar `web.external_layout` se obtiene un encabezado y pie de página con detalles de contacto completos y el logotipo de la empresa disponible. Dado que ambos diseños son vistas, se puede utilizar la herencia usando el `inherit_id` correspondiente.

`< template id =` que debe coincidir con `report_name` y `report_file` de `<record/>` excepto que no lleva el nombre del módulo>

`< t t-call` contenedor (por ejemplo `web.html_container`)>

`< t t-foreach` iteración por modelo>

`< t t-call` llamada a formato por ejemplo `web.internal_layout`>

Entre etiquetas `<div> ... </div>` los campos del modelo que se quieran obtener en el informe. Además, es imprescindible ajustar todo el contenido en un elemento `<page>`.

`(<div class="page"> ... </div>)`

Se pueden crear en uno o varios ficheros `reports.xml` e incluso se pueden separar ambas partes `<record/>` y `<template/>` en ficheros distintos.

Una vez el informe esté creado se obtiene a través de la opción Imprimir. En la url de dicha ventana se puede encontrar el modelo sobre el que está creado.

Se debe poder acceder a los parámetros de configuración `web.base.url` o `report.url` desde la instancia de Odoo, o el informe tardará demasiado en generarse. Para registrar un informe se debe crear un registro `ir.action.report` con una nueva etiqueta.

Las etiquetas todavía son compatibles con Odoo 14 pero el uso de la etiqueta generará una alerta que se registrará (`odoo.log`).



# 7.4.

## Permisos del módulo

Trabajamos creando un fichero security.xml y cambiando el ir.model.access.csv de la carpeta security. Hay que añadir en el fichero xml una línea (record) por cada uno de los roles posibles (grupos), especificando en el fichero csv los permisos que va a tener.

```
<odoo>
    <data>
        <record id="nombredelmodulo.categoriadeusuario model="res.groups">
            <field name="name"> Módulo / Administradora </field>
        <field name="name"> Módulo / Usuario </field>
        ...
        </record>
    </data>
</odoo>
```

El fichero security.csv es de la siguiente forma:

```
id, name,model_id:id,perm_read,perm_write,perm_
create,perm_unlink
```

Teniendo en cuenta que:

- > id: id de la regla
- > name: nombre para la regla
- > model\_id: id: model\_tabla\_ sobre\_ la\_ que\_aplica
- > group\_id: id: grupo al que aplica

Seguidamente se ponen los valores 1 y 0 según si se concede o no ese permiso concreto. Se debe añadir una línea por cada rol añadido en el security.xml creado y por cada modelo.

Después, hay que incluirlo en el \_manifest\_.py (al principio del apartado 'data', primero el security.xml y después el ir.model.access.csv).

Después añadimos en el fichero views.xml el atributo groups="" especificando qué grupo tendrá asignada cada vista y que usuario podrá visualizarlo.

## 7.5.

Creación paso a  
paso de un módulo

Se va a crear un nuevo módulo que se denominará *elbuenlector* y va a permitir gestionar una biblioteca en la que se va a almacenar información acerca de dos objetos: libros y género al que pertenecen.

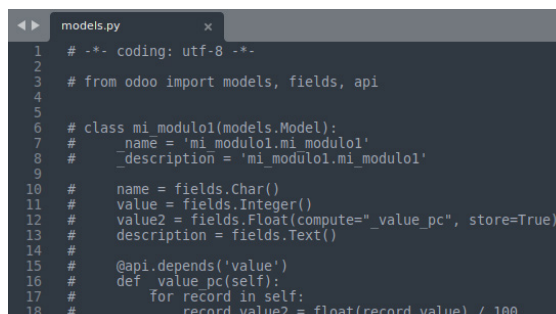
En cuanto al primer objeto, se almacenará la siguiente información: título, autor, año de publicación, número de páginas y una sinopsis.

Por otra parte, el género puede tener los siguientes valores: novela, relato, ensayo o poesía.

Por esto, se trabajará con dos modelos. Los modelos son la base del módulo y se asocian con tables de una base de datos PostgreSQL. Se conoce porque cada modelo en Odoo se extiende de la clase *models.Model* y se definen en uno o varios archivos *models.py* que se guardan en el directorio *models*.

Cada modelo corresponde a una clase Python que se debe definir en *models.py*

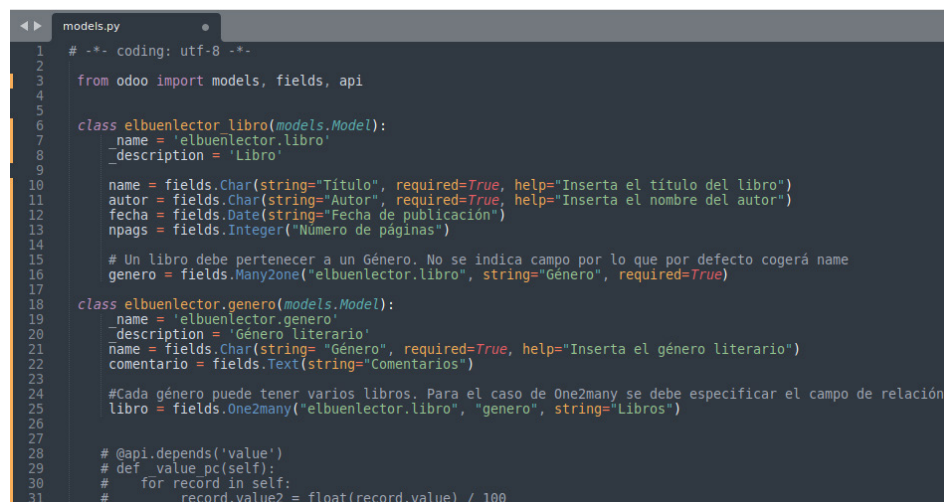
Cuando se abre por primera vez el fichero *models.py* se observa que aparecen una serie de líneas comentadas que dan una idea de lo que se incluye de forma elemental.



```
1 # -*- coding: utf-8 -*-
2
3 # from odoo import models, fields, api
4
5
6 # class mi_modulo1(models.Model):
7 #     _name = 'mi_modulo1.mi_modulo1'
8 #     _description = 'mi_modulo1.mi_modulo1'
9
10 #     name = fields.Char()
11 #     value = fields.Integer()
12 #     value2 = fields.Float(compute='_value_pc', store=True)
13 #     description = fields.Text()
14 #
15 #     @api.depends('value')
16 #     def _value_pc(self):
17 #         for record in self:
18 #             record.value2 = float(record.value) / 100
```

Imagen 5. Estado inicial de models.py

De este modo, unos posibles modelos podrían ser:



```
1 # -*- coding: utf-8 -*-
2
3 from odoo import models, fields, api
4
5
6 class elbuenlector.libro(models.Model):
7     _name = 'elbuenlector.libro'
8     _description = 'Libro'
9
10     name = fields.Char(string="Título", required=True, help="Inserta el título del libro")
11     autor = fields.Char(string="Autor", required=True, help="Inserta el nombre del autor")
12     fecha = fields.Date(string="Fecha de publicación")
13     npags = fields.Integer("Número de páginas")
14
15     # Un libro debe pertenecer a un Género. No se indica campo por lo que por defecto cogerá name
16     genero = fields.Many2one("elbuenlector.genero", string="Género", required=True)
17
18 class elbuenlector.genero(models.Model):
19     _name = 'elbuenlector.genero'
20     _description = 'Género literario'
21     name = fields.Char(string="Género", required=True, help="Inserta el género literario")
22     comentario = fields.Text(string="Comentarios")
23
24     # Cada género puede tener varios libros. Para el caso de One2many se debe especificar el campo de relación
25     libro = fields.One2many("elbuenlector.libro", "genero", string="Libros")
26
27
28 # @api.depends('value')
29 # def _value_pc(self):
30 #     for record in self:
31 #         record.value2 = float(record.value) / 100
```

Imagen 6. Vista de models.py



Se han añadido algunos campos de tipos variados, pero se recomienda profundizar más adelante añadiendo campos de otros tipos y atributos.

Para los campos relacionales entre modelos, se ha hecho una consideración de que un libro pertenece a un género y que un género puede llegar a contener varios libros. Cuando la relación es Many2one no es necesario especificar el campo de relación, ya que por defecto la realiza con *name*.

Una vez se ha reiniciado el servicio de Odoo (paso imprescindible cada vez que se quiera modificar un fichero de Python) y de que se haya instalado el módulo, se podrán encontrar los dos modelos creados en la relación de modelos del ERP que aparece en Ajustes/Modelos.

A continuación, se va a confeccionar Vistas asociadas a los dos modelos. En este caso, al editar el archivo *views.xml* de la carpeta *views* se consigue un código ejemplo que sirve de base al programador.

Los modelos, como ya se sabe, se deben relacionar con las vistas y añadir los elementos de menú y las acciones de ventana. Es en este fichero (y otros ficheros como este que se puedan añadir en un futuro) donde se relacionan.

Toda vista se trata de un objeto de tipo *record* que se basa en *ir.ui.view* que se define unívocamente a través de un *id* que incluye el nombre de módulo, el del modelo y su tipo. Aplicándose al ejemplo, *elbuenlector.libro\_tree* será una vista de tipo árbol (también llamado lista) del modelo libro del módulo *elbuenlector*.

Un código básico para la vista lista del modelo género puede ser:

```

1  <odoo>
2  <data>
3    <record model="ir.ui.view" id="elbuenlector.genero_tree">
4      <field name="name">elbuenlector.genero.tree</field>
5      <field name="model">elbuenlector.genero</field>
6      <field name="arch" type="xml">
7        <tree>
8          <field name="name"/>
9          <field name="comentario"/>
10         </tree>
11       </field>
12     </record>

```

Imagen 7. Código básico vista tree

Y en cuanto a la vista formulario:

```

16  <record model="ir.ui.view" id="elbuenlector.genero_form">
17    <field name="name">elbuenlector.genero.form</field>
18    <field name="model">elbuenlector.genero</field>
19    <field name="arch" type="xml">
20      <form>
21        <group colspan="2" col="2">
22          <field name="name"/>
23          <field name="comentario"/>
24        </group>
25      </form>
26    </field>
27  </record>

```

Imagen 8. Código básico vista formulario

En el código anterior se han añadido instrucciones para que, a continuación de la vista del modelo género, en la parte inferior de la vista, se genere una lista de los libros existentes en este género.



Se vuelve a realizar el proceso para las vistas de tipo árbol y formulario, pero en este caso del objeto libro:

```

30 <odoo>
31 <data>
32 <record model="ir.ui.view" id="elbuenlector.libro tree">
33 <field name="name">elbuenlector.libro.tree</field>
34 <field name="model">elbuenlector.libro</field>
35 <field name="arch" type="xml">
36 <tree>
37 <field name="name"/>
38 <field name="autor"/>
39 <field name="npags"/>
40 </tree>
41 </field>
42 </record>
43
44
45
46 <record model="ir.ui.view" id="elbuenlector.libro form">
47 <field name="name">elbuenlector.libro.form</field>
48 <field name="model">elbuenlector.libro</field>
49 <field name="arch" type="xml">
50 <form>
51 <group colspan="2" col="2">
52 <field name="name"/>
53 <field name="autor"/>
54 <field name="npags"/>
55 <field name="fecha"/>
56 <field name="genero"/>
57 </group>
58 </form>
59 </field>
60 </record>

```

Imagen 9. Vistas del modelo libro

Lo siguiente a realizar son las acciones de ventana de cada modelo, para ello utilizaremos las líneas comentadas del archivo:

```

62 <record model="ir.actions.act_window" id="elbuenlector.genero_action_window">
63 <field name="name">elbuenlector.genero.action_window</field>
64 <field name="res_model">elbuenlector.genero</field>
65 <field name="view_mode">tree,form</field>
66 </record>
67
68 <record model="ir.actions.act_window" id="elbuenlector.libro_action_window">
69 <field name="name">elbuenlector.libro.action_window</field>
70 <field name="res_model">elbuenlector.libro</field>
71 <field name="view_mode">tree,form</field>
72 </record>

```

Imagen 10. Acciones de ventana

Y, para finalizar, los elementos de menú:

```

75 <menuitem name="ElBuenLector" id="elbuenlector.menu_root"/>
76
77 <menuitem name="Género" id="elbuenlector.genero_menu" parent="elbuenlector.menu_root" action="
78 elbuenlector.genero_action_window"/>
79 <menuitem name="Libro" id="elbuenlector.libro_menu" parent="elbuenlector.menu_root" action="
80 elbuenlector.libro_action_window"/>
81 </data>
82 </odoo>

```

Imagen 11. Elementos de menú

Se procede a instalar el módulo de elbuenlector, y con ello nos aparecerá como módulos instalados y podremos acceder a su vista general, a la vista de formulario y a la vista árbol de ambos modelos.

El proceso para generar informes ya se ha observado en el apartado 7.3. Un ejemplo de informe de la aplicación que se ha desarrollado podría ser:



```

1  <?xml version="1.0" encoding="utf-8"?>
2  <odoo>
3      <report
4          id="elbuenlector.informe_libro"
5          string="Informe sobre libros"
6          model="elbuenlector.libro"
7          report_type="qweb-pdf"
8          name="elbuenlector.informe_libro_view"
9          file="elbuenlector.informe_libro_view"/>
10
11      <template id="informe_libro_view">
12          <t t-call="web.basic_layout">
13              <t t-foreach="docs" t-as="libro">
14                  <div class="page">
15                      <h1 t-field="libro.name"/>
16                      <div>
17                          <strong>>Autor; </strong>
18                          <span t-field="libro.autor"/>
19                      </div>
20                      <div>
21                          <strong>>Nº páginas: </strong>
22                          <span t-field="libro.npages"/>
23                      </div>
24                      <div>
25                          <strong>>Género: </strong>
26                          <span t-field="libro.genero"/>
27                      </div>
28                  </div>
29              </t>
30          </t>
31      </template>
32  </odoo>

```

Imagen 12. Ejemplo de informe

Para obtener el informe es necesario que se incluya la ruta del fichero xml del informe en el fichero `__manifest__.py`, como se muestra a continuación:

```

21  # any module necessary for this one to work correctly
22  'depends': ['base'],
23
24  # always loaded
25  'data': [
26      # 'security/ir.model.access.csv',
27      'views/views.xml',
28      'views/templates.xml',
29      'reports/informe_libro.xml'
30  ],
31  # only loaded in demonstration mode
32  'demo': [
33      'demo/demo.xml',
34  ],
35  }

```

Imagen 13. Inclusión en `__manifest__.py`

Hasta este momento se ha trabajado con el superusuario, de modo que se han evitado incidencias con los permisos, pero si se inicia sesión con otro tipo de usuario, incluso si se trata del administrador, se podrá comprobar que no aparece el nuevo módulo de elbuenlector. Esto se debe a que todavía no se han asignado los permisos pertinentes del módulo a los grupos de usuarios que van a poder utilizarlo.

Una vez se tengan creados los distintos grupos de usuarios del módulo (por ejemplo, usuario, coordinador, jefe, administrador, etc.) se tienen que asignar las distintas vistas a los grupos adecuados (o dejarlas visibles para todos los grupos) y añadir estos usuarios del ERP a los grupos pertinentes.

Para ello es necesario crear el archivo `security.xml` en el directorio `security` con un registro (`<record>`) por cada grupo que se haya creado, con el formato siguiente:



```
security.xml
1 <?xml version="1.0" encoding="utf-8"?>
2 <odoo>
3   <record id="elbuenlector_coordinador" model="res.groups">
4     <field name="name">ElBuenLector / Coordinador</field>
5   </record>
6   <record id="elbuenlector_jefe" model="res.groups">
7     <field name="name">ElbuenLector / Jefe</field>
8   </record>
9 </odoo>
```

Imagen 14. Archivo security.xml

Y, con el archivo creado, se le añaden los permisos específicos a cada grupo en cada modelo en el *ir.model.access.csv*

```
ir.model.access.csv
1 id,name,model id:id,group id:id,perm read,perm write,perm create,perm unlink
2 elbuenlector_genero_jefe,Genero Jefe,model_elbuenlector_genero,elbuenlector_jefe,1,1,1,1
3 elbuenlector_libro_jefe,Libro Jefe,model_elbuenlector_libro,elbuenlector_jefe,1,1,1,1
4 elbuenlector_genero_coordinador,Genero coordinador,model_elbuenlector_genero,elbuenlector_coordinador,1,0,0,0
5 elbuenlector_libro_coordinador,Libro Coordinador,model_elbuenlector_libro,lapeliculera_coordinador,1,0,0,0
```

Imagen 15. Archivo ir.model.access.csv con permisos

Como indica el encabezado del archivo, los cuatro dígitos corresponden a los permisos de lectura, escritura, creación y eliminación.

Los dos archivos, el de *security.xml* y el *ir.model.access.csv* se añaden al *\_\_manifest\_\_.py* en el orden correcto, ya que de otro modo se pueden generar errores por hacer referencia a objetos que aún no han sido definidos.

```
24 # always loaded
25 'data': [
26   'security/security.xml',
27   'security/ir.model.access.csv',
28   'views/views.xml',
29   'views/templates.xml',
30   'reports/informe_libro.xml'
31 ],
```

Imagen 16. Archivo \_\_manifest\_\_.py con los archivos añadidos

De este modo, cuando se inicie sesión con un usuario distinto al superusuario, se mostrará el módulo elbuenlector y en los Ajustes de usuario de cualquier usuario se podrá asignar a qué grupo de los creados pertenece.

En el archivo *views.xml* se pueden asignar los elementos de menú creados a un grupo específico para que solo pueda ser accesible para ese grupo. Y en el caso de que no tenga ningún grupo asignado será accesible para todos, y en función de los permisos que tenga cada usuario que acceda tendrá disponible unas funcionalidades distintas.

Si se desea que un módulo tenga un icono identificativo, se debe subir un fichero *.png* con el nombre de *icon.png* al directorio */static/description* que hay que crear en el directorio principal del módulo.



 [www.universae.com](http://www.universae.com)

