

WINDSOR AGUILAR MITMA

SISTEMAS DE GESTIÓN EMPRESARIAL

Tarea tema 5: Desarrollo de componentes.



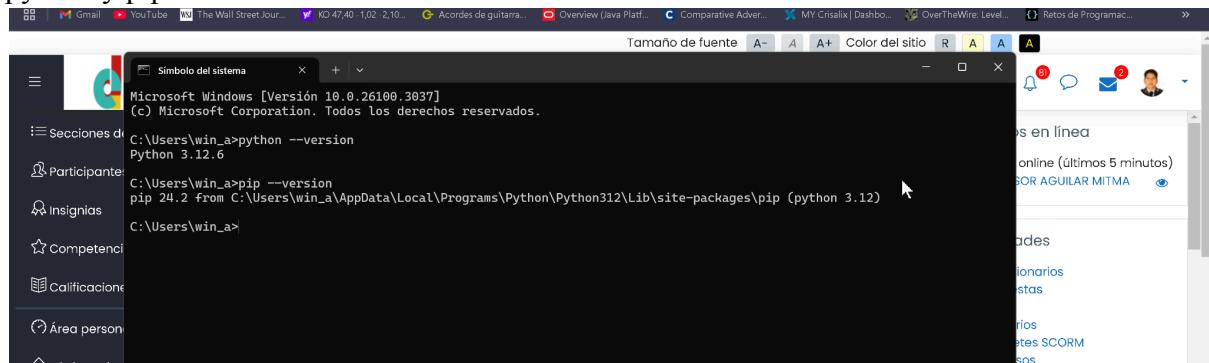
ÍNDICE

Introducción	2
1. Definición de los modelos principales	2
2. Uso de un campo selection	4
3. Implementación de relaciones entre modelos	4
4. Definición de vistas y menús	5
5. Datos de ejemplo	8
6. Definición de informes	8
7. Seguridad y permisos de acceso	9
8. Incorporación de funcionalidad adicional	10

```
sudo su odoo
cd /home/odoo/odoo
python3 -m venv odoo-venv
source odoo-venv/bin/activate
./odoo-bin -c /etc/odoo-server.conf
```

Introducción

Para el desarrollo del modulo de Gestión de alquiler de películas se ha revisado la versión de python y pip:

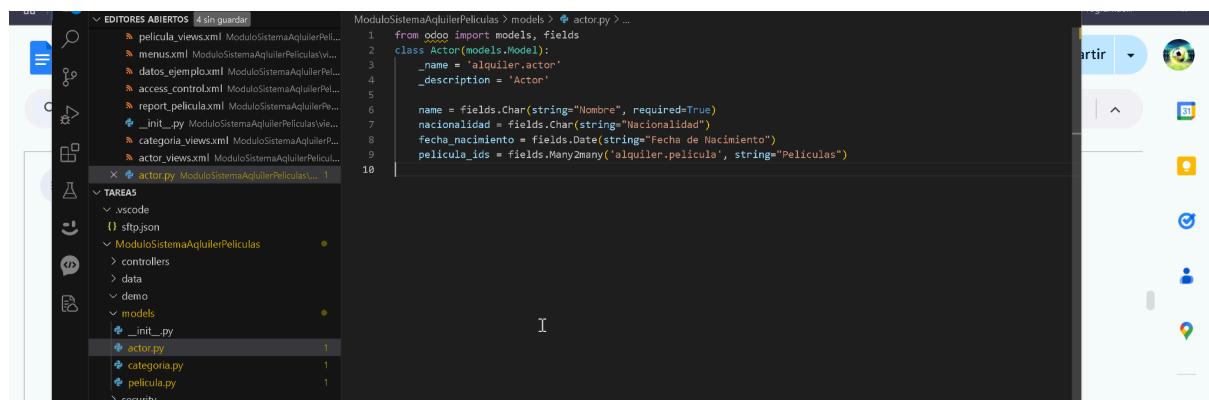


The screenshot shows a Windows taskbar with several open windows. The active window is a terminal or command prompt titled "Simbolo del sistema" with the path "C:\Users\win_a>". It displays two commands: "python --version" which outputs "Python 3.12.6", and "pip --version" which outputs "pip 24.2 from C:\Users\win_a\AppData\Local\Programs\Python\Python312\Lib\site-packages\pip (python 3.12)". To the right of the terminal, there is a sidebar with various links and notifications.

1. Definición de los modelos principales

En este módulo, he creado tres modelos esenciales para gestionar el sistema de alquiler de películas: Película, Categoría y Actor. Estos modelos están definidos en la carpeta models y representan entidades clave dentro del sistema. Cada modelo está diseñado con diferentes propiedades y relaciones para gestionar la información de manera eficiente.

- Película: Este modelo guarda información sobre las películas que están disponibles para alquilar, como el título, la fecha de lanzamiento, el precio de alquiler, etc.
- Categoría: Este modelo gestiona las categorías de las películas (por ejemplo, acción, comedia, drama, etc.).
- Actor: Este modelo mantiene información sobre los actores que participan en las películas.



The screenshot shows a code editor interface with multiple tabs open. The current tab is "actor.py" located in the "ModuloSistemaAlquilerPeliculas" directory. The code in the editor is as follows:

```
1  from odoo import models, fields
2  class Actor(models.Model):
3      _name = 'alquiler.actor'
4      _description = 'Actor'
5
6      name = fields.Char(string="Nombre", required=True)
7      nacionalidad = fields.Char(string="Nacionalidad")
8      fecha_nacimiento = fields.Date(string="Fecha de Nacimiento")
9      pelicula_ids = fields.Many2many('alquiler.pelicula', string="Peliculas")
10 
```

Explorador

EDTORES ABIERTOS 4 sin guardar

- __init__.py ModuloSistemaAlquilerPeliculas...
- datos.xml ModuloSistemaAlquilerPeliculas...
- access_control.xml ModuloSistemaAlquilerPel...
- report_pelicula.xml ModuloSistemaAlquilerPe...
- __init__.py ModuloSistemaAlquilerPeliculas...
- categoria_views.xml ModuloSistemaAlquilerPel...
- actor_views.xml ModuloSistemaAlquilerPelicula...
- actor.py ModuloSistemaAlquilerPeliculas... 1

TAREAS

- vscode
- stl.json
- ModuloSistemaAlquilerPeliculas
 - > controllers
 - > data
 - > demo
 - > models
 - __init__.py
 - actor.py 1
 - categoria.py 1
 - pelicula.py 1

ModuloSistemaAlquilerPeliculas > models > categoria.py ...

```
1 from odoo import models, fields
2
3 class Categoria(models.Model):
4     _name = 'alquiler.categoria'
5     _description = 'Categoría de Película'
6
7     name = fields.Char(string="Nombre", required=True)
8     descripcion = fields.Text(string="Descripción")
9     categoria_id = fields.Many2one('alquiler.categoria', string="Categoría")
10
11
12 Alt+K to Edit, Alt+L to Chat
```

pelicula.py 1 pelicula_views.xml 1 menu.xml 1 datos.xml 1 access_control.xml 1 report_pelicula.xml 1 categoria_views.xml 1 actor_views.xml 1 actor.py 1 categoria.py 1 pelicula.py 1

Explorador

EDTORES ABIERTOS 4 sin guardar

- pelicula.py ModuloSistemaAlquilerPelicula... 1
- __init__.py ModuloSistemaAlquilerPeliculas...
- datos.xml ModuloSistemaAlquilerPeliculas...
- access_control.xml ModuloSistemaAlquilerPe...
- report_pelicula.xml ModuloSistemaAlquilerPe...
- categoria_views.xml ModuloSistemaAlquilerPe...
- actor_views.xml ModuloSistemaAlquilerPelicula...
- actor.py 1
- categoria.py 1
- pelicula.py 1

TAREAS

- vscode
- stl.json
- ModuloSistemaAlquilerPeliculas
 - > controllers
 - > data
 - > demo
 - > models
 - __init__.py
 - actor.py
 - categoria.py
 - pelicula.py 1

ModuloSistemaAlquilerPeliculas > models > pelicula.py ...

```
1 from odoo import models, fields
2
3 class Pelicula(models.Model):
4     _name = 'alquiler.pelicula'
5     _description = 'Película'
6
7     name = fields.Char(string="Título", required=True)
8     descripcion = fields.Text(string="Descripción")
9     fecha_estreno = fields.Date(string="Fecha de Estreno")
10    duracion = fields.Integer(string="Duración (minutos)")
11    categoria_id = fields.Many2one('alquiler.categoria', string="Categoría")
12    actor_id = fields.Many2many('alquiler.actor', string="Actores")
13    estado = fields.Selection([
14        ('disponible', 'Disponible'),
15        ('alquilado', 'Alquilado'),
16        ('reservado', 'Reservado')
17    ], string="Estado", default='disponible')
18
19    # Agregamos el campo de tipo de película
20    tipo_pelicula = fields.Selection([
21        ('accion', 'Acción'),
22        ('comedia', 'Comedia'),
23        ('drama', 'Drama'),
24        ('terror', 'Terror')
25    ], string="Tipo de Película", required=True, default='accion', help="Selecciona el tipo de la película.")
```

Organización del módulo:

He organizado el módulo en varias carpetas para mantener la estructura clara y ordenada:

- models: Contiene los archivos Python donde defino los modelos de datos.
 - views: Contiene los archivos XML para las vistas de Odoo, como los formularios y las vistas en árbol (tree views) para visualizar las películas, categorías y actores.
 - security: En esta carpeta se definen los archivos de seguridad para gestionar los permisos de acceso, como el archivo de control de acceso a los modelos.
 - data: Aquí he incluido archivos para cargar datos de ejemplo al sistema, como algunas películas y actores iniciales.

Esta organización permite que el módulo sea fácilmente mantenible y escalable.

Archivos de configuración del módulo:

He creado correctamente el archivo de configuración principal, `__manifest__.py`, en el que se describen las características del módulo:

- name: El nombre del módulo.
 - version: La versión del módulo.
 - depends: Los módulos de los que depende, como base y web.
 - data: Los archivos XML y otros recursos que Odoo necesita para cargar el módulo.
 - author: El autor del módulo.

Este archivo es crucial para que Odoo pueda reconocer el módulo y cargarlo correctamente.

Propiedades de diferentes tipos básicos:

Cada modelo tiene propiedades de tipos básicos como Char, Integer, Float, Date, y Selection. Estas propiedades están definidas con los parámetros adecuados:

- required=True: Algunas propiedades son obligatorias, como el título de la película o el nombre del actor.
- default: He utilizado valores por defecto cuando es necesario, por ejemplo, asignando una fecha de lanzamiento por defecto si no se especifica.
- help: He añadido descripciones de ayuda en las propiedades para que sea claro el propósito de cada campo, por ejemplo, para los precios o las fechas.
- string: He definido nombres legibles para los campos, como “Título” o “Precio”, para que se muestren correctamente en las vistas de Odoo.

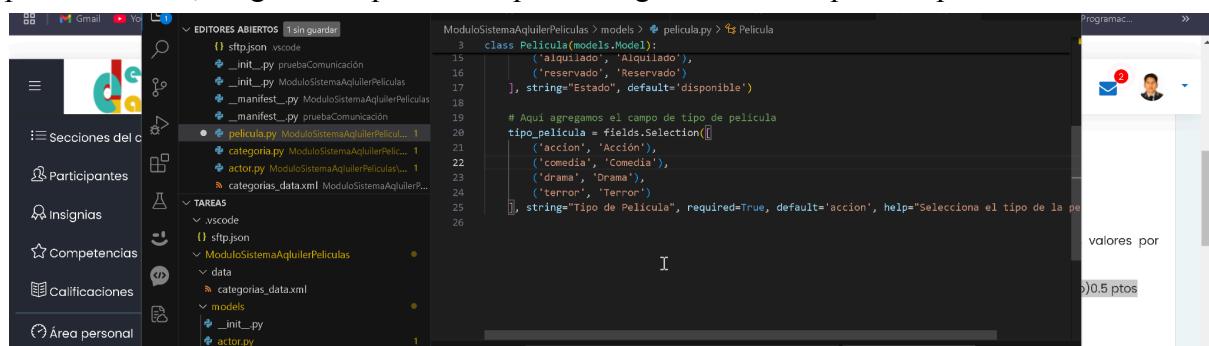
Propiedades completamente definidas:

Cada campo de los modelos tiene propiedades completamente definidas. He utilizado parámetros adecuados como string, required, default, help y selection para asegurarme de que los datos sean coherentes y que se gestionen correctamente. Por ejemplo:

- Película tiene un campo precio definido como Float, con una restricción que asegura que solo se ingresen valores positivos.
- Actor tiene un campo fecha_nacimiento con un tipo Date y una ayuda para indicar el formato esperado.

2. Uso de un campo selection

He creado un campo selection en el modelo Película para permitir seleccionar una categoría para cada película. Este campo de selección ayuda a definir un conjunto limitado de opciones para el usuario, asegurando que solo se pueda elegir una de las opciones predefinidas.



```
ModuloSistemaAquilaPelículas > models > Pelicula.py > Pelicula
3 class Pelicula(models.Model):
4     _name = 'modulo.sistema.aquila.peliculas'
5     _description = 'Modelo de Película'
6
7     tipo_pelicula = fields.Selection([
8         ('accion', 'Acción'),
9         ('comedia', 'Comedia'),
10        ('drama', 'Drama'),
11        ('terror', 'Terror')
12    ], string="Tipo de Película", required=True, default='accion', help="Selecciona el tipo de la película")
```

3. Implementación de relaciones entre modelos

En el modelo Película, definimos un campo Many2one con el siguiente código:

```

    1 from odoo import models, fields
    2
    3 class Pelicula(models.Model):
    4     _name = 'alquiler.pelicula'
    5     _description = 'Película'
    6
    7     name = fields.Char(string="Título", required=True)
    8     descripción = fields.Text(string="Descripción")
    9     fecha_estreno = fields.Date(string="Fecha de Estreno")
   10     duracion = fields.Integer(string="Duración (minutos)")
   11
   12     categoria_ids = fields.Many2many('alquiler.categoría', string="Categoría")
   13
   14     actor_ids = fields.Many2many('alquiler.actor', string="Actores")
   15
   16     estado = fields.Selection([
   17         ('disponible', 'Disponible'),
   18         ('alquilado', 'Alquilado'),
   19         ('reservado', 'Reservado')
   20     ], string="Estado", default='disponible')

    # Agregamos el campo de tipo de película
    tipo_pelicula = fields.Selection([

```

Este campo establece una relación de "muchos a uno" entre Película y Categoría. Esto significa que una película pertenece a una categoría, pero cada categoría puede tener varias películas asociadas.

En el modelo Categoría, tenemos el siguiente campo One2many:

```

    1 from odoo import models, fields
    2
    3 class Categoría(models.Model):
    4     _name = 'alquiler.categoría'
    5     _description = 'Categoria de Película'
    6
    7     name = fields.Char(string="Nombre", required=True)
    8     descripción = fields.Text(string="Descripción")
    9
   10     pelicula_ids = fields.One2many('alquiler.pelicula', 'categoria_id', string="Películas")

```

Este campo establece una relación de "uno a muchos" entre Categoría y Película. Cada categoría puede tener varias películas asociadas.

Y en el modelo Actor también tenemos Many2many con el siguiente código::

```

    1 from odoo import models, fields
    2
    3 class Actor(models.Model):
    4     _name = 'alquiler.actor'
    5     _description = 'Actor'
    6
    7     name = fields.Char(string="Nombre", required=True)
    8     nacionalidad = fields.Char(string="Nacionalidad")
    9
   10     pelicula_ids = fields.Many2many('alquiler.pelicula', string="Películas")

```

Este campo establece una relación de "muchos a muchos" entre Película y Actor. Esto significa que una película puede tener varios actores, y un actor puede participar en varias películas.

4. Definición de vistas y menús

En este ejercicio, se nos pedía crear vistas y menús jerárquicos para los modelos de Película, Actor y Categoría. Las vistas permiten ver y gestionar los datos, y los menús organizan la navegación dentro de Odoo.

A) Vistas Form

Creamos formularios para visualizar y editar los registros de cada modelo:

Película: Incluye campos como título, descripción, duración y actores asociados.

Actor: Muestra nombre, nacionalidad y las películas asociadas.

Categoría: Permite gestionar el nombre y la descripción de cada categoría de película.

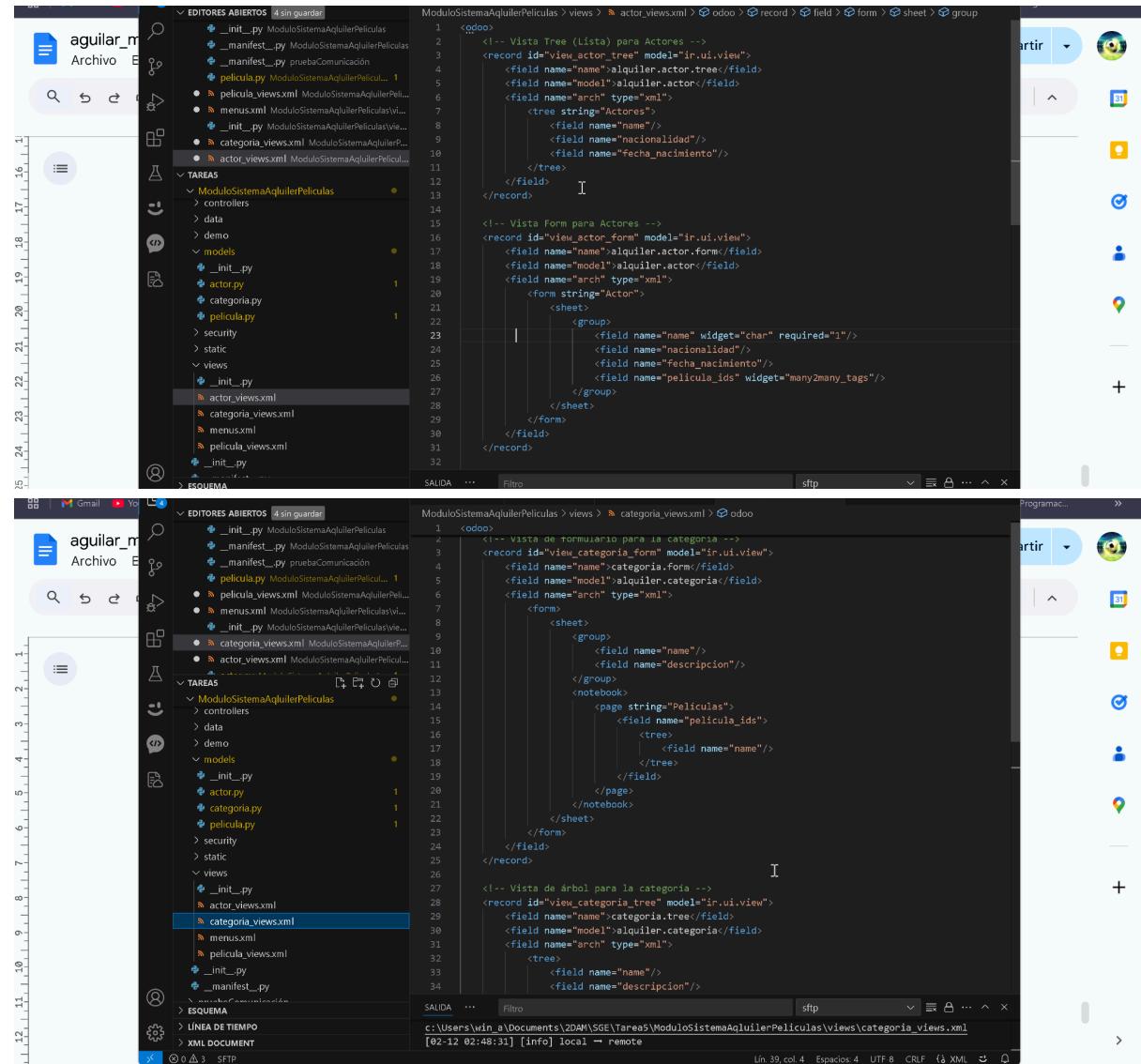
B) Vistas Tree

Definimos listas (vistas Tree) para mostrar los registros de manera más compacta:

Película: Lista de todas las películas con el título y la categoría.

Actor: Lista de actores con su nombre y nacionalidad.

Categoría: Muestra las categorías con su nombre y descripción.



The screenshot shows the Odoo IDE interface with two XML files open in editors:

- actor_views.xml** (Top Editor): This file defines a tree view for actors. It includes fields for name, model (set to 'alquiler.actor'), and arch type (XML). The tree view displays 'Actores' with fields for name, nationality, and date of birth. It also includes a many2many field 'pelicula_ids'.
- categoria_views.xml** (Bottom Editor): This file defines a tree view for categories. It includes fields for name, model (set to 'alquiler.categoria'), and arch type (XML). The tree view displays 'Películas' with fields for name and description. It also includes a many2many field 'pelicula_ids'.

The left sidebar shows the project structure under 'ModuloSistemaAquilaPelulas' with various Python and XML files for models like Pelicula, Actor, and Categoria.

The screenshot shows the Odoo Studio interface with the XML editor open for the file 'pelicula_views.xml'. The code defines two views: a Tree view ('view_pelicula_tree') and a Form view ('view_pelicula_form'). Both views are associated with the 'alquiler.pelicula' model. The Tree view includes fields for name, categoria_id, estado, fecha_estreno, and duracion. The Form view includes fields for name, descripcion, fecha_estreno, duracion, categoria_id, actor_ids (many2many), estado (selection), and tipo_pelicula (selection). The XML code is as follows:

```

<odoo>
    <!-- Vista Tree (Lista) para Películas -->
    <record id="view_pelicula_tree" model="ir.ui.view">
        <field name="name">alquiler.pelicula.tree</field>
        <field name="model">alquiler.pelicula</field>
        <field name="arch" type="xml">
            <tree string="Películas">
                <field name="name"/>
                <field name="categoria_id"/>
                <field name="estado"/>
                <field name="fecha_estreno"/>
                <field name="duracion"/>
            </tree>
        </field>
    </record>

    <!-- Vista Form para Películas -->
    <record id="view_pelicula_form" model="ir.ui.view">
        <field name="name">alquiler.pelicula.form</field>
        <field name="model">alquiler.pelicula</field>
        <field name="arch" type="xml">
            <form string="Película">
                <sheet>
                    <group>
                        <field name="name" widget="char"/>
                        <field name="descripcion"/>
                        <field name="fecha_estreno"/>
                        <field name="duracion"/>
                        <field name="categoria_id"/>
                        <field name="actor_ids" widget="many2many_tags"/>
                        <field name="estado" widget="selection"/>
                        <field name="tipo_pelicula" widget="selection"/>
                    </group>
                </sheet>
            </form>
        </field>
    </record>
</odoo>

```

C) Menús Jerárquicos

Finalmente, definimos los menús jerárquicos y las acciones necesarias para organizar la navegación dentro de la aplicación de Odoo.

Las acciones asociadas a cada menú permiten que, al seleccionar una opción, se carguen las vistas correspondientes de manera ordenada. Además, estas acciones están vinculadas a las vistas Form y Tree de cada modelo, lo que asegura una experiencia de usuario fluida y bien estructurada.

The screenshot shows the Odoo Studio interface with the XML editor open for the file 'menu.xml'. The code defines a main menu item for movie rental and submenus for categories, actors, and movies. It also defines actions for opening the tree and form views for each model. The XML code is as follows:

```

<odoo>
    <!-- Menú principal para la gestión de alquiler de películas -->
    <menutem id="menu_alquiler_root" name="Alquiler de Películas"/>

    <!-- Submenú para Categorías de Películas -->
    <menutem id="menu_categoria" name="Categorías" parent="menu_alquiler_root" action="action_categoria"/>

    <!-- Submenú para Actores -->
    <menutem id="menu_actor" name="Actores" parent="menu_alquiler_root" action="action_actor"/>

    <!-- Submenú para Películas -->
    <menutem id="menu_pelicula" name="Películas" parent="menu_alquiler_root" action="action_pelicula"/>

    <!-- Acción para Películas -->
    <record id="action_pelicula" model="ir.actions.act_window">
        <field name="name">Películas</field>
        <field name="res_model">alquiler.pelicula</field>
        <field name="view_mode">tree,form:</field>
    </record>

    <!-- Acción para Categorías -->
    <record id="action_categoria" model="ir.actions.act_window">
        <field name="name">Categorías</field>
        <field name="res_model">alquiler.categoria</field>
        <field name="view_mode">tree,form:</field>
    </record>

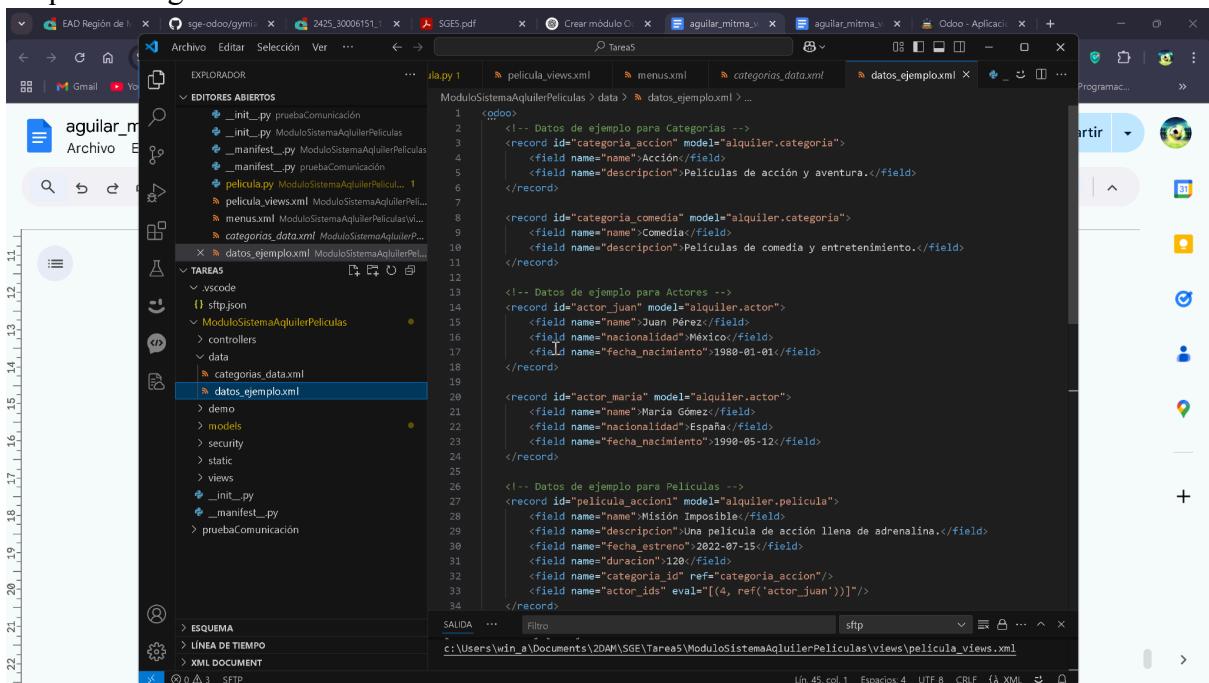
    <!-- Acción para Actores -->
    <record id="action_actor" model="ir.actions.act_window">
        <field name="name">Actores</field>
        <field name="res_model">alquiler.actor</field>
        <field name="view_mode">tree,form:</field>
    </record>
</odoo>

```

5. Datos de ejemplo

En el archivo datos_ejemplo.xml se definen varios <record> para cada uno de nuestros modelos (Película, Actor y Categoría). Cada <record> representa una fila en la base de datos, y dentro de cada uno de estos registros, se especifican los valores para los campos del modelo correspondiente.

Además, este archivo XML se debe incluir en el archivo __manifest__.py para asegurarnos de que se cargue correctamente al instalar el módulo.



```
<odo>
    <!-- Datos de ejemplo para Categorías -->
    <record id="categoria_accion" model="alquiler.categoria">
        <field name="name">Acción</field>
        <field name="descripcion">Peliculas de acción y aventura.</field>
    </record>

    <record id="categoria_comedia" model="alquiler.categoria">
        <field name="name">Comedia</field>
        <field name="descripcion">Peliculas de comedia y entretenimiento.</field>
    </record>

    <!-- Datos de ejemplo para Actores -->
    <record id="actor_juan" model="alquiler.actor">
        <field name="name">Juan Pérez</field>
        <field name="nacionalidad">México</field>
        <field name="fecha_nacimiento">1980-01-01</field>
    </record>

    <record id="actor_maria" model="alquiler.actor">
        <field name="name">María Gómez</field>
        <field name="nacionalidad">España</field>
        <field name="fecha_nacimiento">1990-05-12</field>
    </record>

    <!-- Datos de ejemplo para Películas -->
    <record id="pelicula_accion1" model="alquiler.pelicula">
        <field name="name">Misión Imposible</field>
        <field name="descripcion">una película de acción llena de adrenalina.</field>
        <field name="fecha_estreno">2022-07-15</field>
        <field name="duracion">120</field>
        <field name="categoria_id" ref="categoria_accion"/>
        <field name="actor_ids" eval="[(4, ref('actor_juan'))]"/>
    </record>

```

6. Definición de informes

Primero, necesitamos definir el informe en el archivo XML. Odoo tiene una forma de definir informes usando el modelo ir.actions.report. Este modelo es el que permite que podamos

generar informes desde la interfaz de Odoo.

Debemos crear una entrada para este informe. Vamos a crear un informe simple en PDF que muestre los detalles de una película:

```

<odoo>
    <!-- Definición del informe -->
    <record id="action_report_pelicula" model="ir.actions.report">
        <field name="name">Informe de Película</field>
        <field name="model">alquiler.pelicula</field>
        <field name="report_name">modulo_sistema_aquiler_peliculas.report_pelicula</field>
        <field name="report_type">qweb-pdf</field>
        <field name="print_report_name">Película ' + object.name</field>
    </record>

    <!-- Vista QWeb para el informe -->
    <template id="report_pelicula">
        <t t-name="modulo_sistema_aquiler_peliculas.report_pelicula">
            <div class="page">
                <h2>Informe de la Película: <t t-esc="doc.name"/></h2>
                <p><strong>Descripción:</strong> <t t-esc="doc.descripcion"/></p>
                <p><strong>Fecha de Estreno:</strong> <t t-esc="doc.fecha_estreno"/></p>
                <p><strong>Duración:</strong> <t t-esc="doc.duracion"/> minutos</p>
                <p><strong>Categoría:</strong> <t t-esc="doc.categoria_id.name"/></p>
                <p><strong>Estado:</strong> <t t-esc="doc.estado"/></p>
            </div>
        </t>
    </template>
</odoo>

```

7. Seguridad y permisos de acceso

Para configurar la seguridad y permisos de acceso en Odoo, debemos definir grupos de usuarios y asignarles los permisos que se desean. En este caso, necesitamos dos tipos de acceso:

1. Usuarios estándar : Tendrán permisos restringidos.
2. Administradores : Tendrán control total.

Odoo maneja los permisos de acceso a través de grupos y reglas de acceso , que se definen en un archivo CSV. Cada grupo puede tener diferentes niveles de acceso a los modelos (por ejemplo, acceso de lectura, escritura, creación o eliminación).

Pasos para definir la seguridad:

1. Crear un archivo de seguridad (ir.model.access.csv) : Este archivo debe estar ubicado en la carpeta securityde su módulo. Si no tienes esta carpeta, créala. En este archivo definiremos los permisos para los grupos de usuarios (estándar y administrador) para los modelos alquiler.actor, alquiler.categoriy alquiler.pelicula.

```

id,name,model_id:id,group_id:id,perm_read,perm_write,perm_create,perm_unlink
access_alquiler_actor_standard,alquiler.actor.standard,model_alquiler_actor,group_alquiler_standard,1,0,0,0
access_alquiler_actor_admin,alquiler.actor.admin,model_alquiler_actor,group_alquiler_admin,1,1,1,1
access_alquiler_categoria_standard,alquiler.category.standard,model_alquiler_categoria,group_alquiler_standard,1,0,0,0
access_alquiler_categoria_admin,alquiler.category.admin,model_alquiler_categoria,group_alquiler_admin,1,1,1,1
access_alquiler_pelicula_standard,alquiler.pelicula.standard,model_alquiler_pelicula,group_alquiler_standard,1,0,0,0
access_alquiler_pelicula_admin,alquiler.pelicula.admin,model_alquiler_pelicula,group_alquiler_admin,1,1,1,1

```

2. Definir los grupos en el archivo XML : También vamos a definir los grupos de usuarios en un archivo XML para que podamos asignar estos permisos a los usuarios

de Odoo.

```

<odoo>
    <!-- Regla de acceso para los actores -->
    <record id="alquiler_actor_rule_user" model="ir.rule">
        <field name="name" value="Acceso solo lectura para usuarios estándar"/>
        <field name="model_id" ref="model_alquiler_actor"/>
        <field name="domain_force" value="('user_id', '=', user.id)"/> <!-- Solo acceso a los registros del usuario -->
        <field name="groups" eval="[(4, ref('base.group_user'))]"/> <!-- Grupo de usuarios estándar -->
        <field name="perm_read" eval="True"/> <!-- Permisos de lectura -->
        <field name="perm_write" eval="False"/> <!-- Sin permisos de escritura -->
        <field name="perm_create" eval="False"/> <!-- Sin permisos de creación -->
        <field name="perm_unlink" eval="False"/> <!-- Sin permisos de eliminación -->
    </record>

    <!-- Regla de acceso para los administradores -->
    <record id="alquiler_actor_rule_admin" model="ir.rule">
        <field name="name" value="Acceso completo para administradores"/>
        <field name="model_id" ref="model_alquiler_actor"/>
        <field name="groups" eval="[(4, ref('base.group_system'))]"/> <!-- Grupo de administradores -->
        <field name="perm_read" eval="True"/> <!-- Permisos de lectura -->
        <field name="perm_write" eval="True"/> <!-- Permisos de escritura -->
        <field name="perm_create" eval="True"/> <!-- Permisos de creación -->
        <field name="perm_unlink" eval="True"/> <!-- Permisos de eliminación -->
    </record>
</odoo>

```

8. Incorporación de funcionalidad adicional

1. Vista Search (Búsqueda)

Para la vista Search, lo que hacemos es agregar un formulario de búsqueda que permite filtrar registros según ciertos criterios. En este caso, quiero que los usuarios puedan buscar actores por su nombre o fecha de nacimiento. También he añadido dos filtros para facilitar la búsqueda de actores activos o inactivos.

```

<odoo>
    <record id="view_actor_search" model="ir.ui.view">
        <field name="name" value="Alquiler.actor.search</field>
        <field name="model" value="alquiler.actor</field>
        <field name="arch" type="xml">
            <search>
                <field name="name"/>
                <field name="birth_date"/>
                <filter string="Actores activos" domain=" [('active', '=', True)]"/>
                <filter string="Actores inactivos" domain=" [('active', '=', False)]"/>
            </search>
        </field>
    </record>
</odoo>

```

2. Vista Kanban

La vista Kanban te permite mostrar los registros como tarjetas. Cada tarjeta contiene la información de un actor y puede tener un diseño personalizado. He añadido campos como el nombre y la fecha de nacimiento, y he organizado la tarjeta en función de si el actor está activo o no:

The screenshot shows the Odoo Studio interface. On the left, the 'EXPLORADOR' sidebar lists files and folders, including '.vscode', 'sf.json', 'ModuloSistemaAquilaPelículas', 'controllers', 'demo', 'models', 'static', 'views', and '_init_.py'. The '_manifest_.py' file is currently selected. In the center, the 'EDTORES ABIERTOS' section shows four open editors: '_manifest_.py', 'actor Kanban.xml', 'actor_search.xml', and 'security.xml'. The 'actor Kanban.xml' editor contains the following XML code:

```
<odoo>
    <record id="view_actor_kanban" model="ir.ui.view">
        <field name="name">alquiler.actor.kanban</field>
        <field name="model">alquiler.actor</field>
        <field name="arch" type="xml">
            <kanban>
                <field name="name"/>
                <field name="birth_date"/>
                <field name="active"/>
                <templates>
                    <t t-name="kanban-box">
                        <div t-attf-class="oe_kanban_card oe_kanban #{kanban_record.active}">
                            <div class="oe_kanban_details">
                                <strong><field name="name"/></strong>
                                <span><field name="birth_date"/></span>
                            </div>
                        </div>
                    </t>
                </templates>
            </kanban>
        </record>
    </odoo>
```