Documentación Técnica: Modelos y vistas de los módulos de Odoo

Autor: Pedro José Meixús Belsol

15 de diciembre de 2025

Índice

1. Modificar lista de tareas 2

2. Ampliación módulo biblioteca de cómics 5

3. Creación módulo de un hospital 10

4. Creación de módulo de ciclo formativo 15

1. Modificar lista de tareas

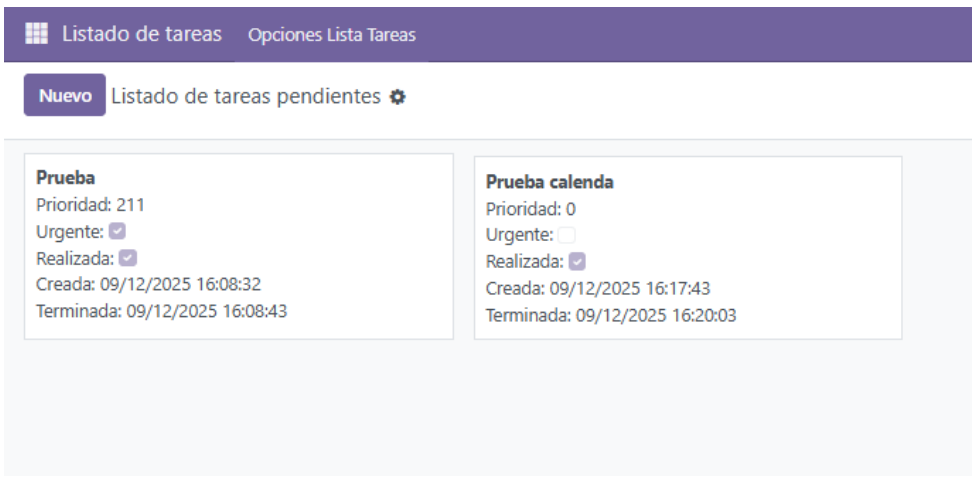
En esta primera actividad vamos a modificar la lista de tareas básica, en mi caso utilizaré la que habia modificado levemente para la anterior práctica (incluye una fecha de inicio y otra de fin).

Lo primero que haremos será cambiar nuestra lista de tareas por una vista Kanban, para ello simplemente vamos a cambiar todo lo relacionado con "list"por "kanbanz añadiremos un bloque de código "template"para indicar como se debe mostrar cada tarjeta del kanban. Quedaría de la siguiente manera:

```
<templates>
    <t t-name="card">
        <div class="oe_kanban_card">
            <strong><field name="tarea"/></strong>
            <div>Prioridad: <field name="prioridad"/></div>
            <div>Urgente: <field name="urgente"/></div>
            <div>Realizada: <field name="realizada"/></div>
            <div>Creada: <field name="fecha_creacion"/></div>
            <div>Terminada: <field name="fecha_terminada"/></div>
        </div>
    </t>
</templates>
```

Código 1: Template para vista Kanban

La vista deberia quedar algo similar a esto.



Ahora nos falta añadir una vista de tipo calendario, donde usaré las fechas que ya cree en práctica anterior. Tendremos que añadir un nuevo recordz dentro de este meteremos un çalendar"donde le indicaremos las fechas de inicio y fin (obtenidas de los campos correspondientes), el color si es prioritaria y el nombre.

```xml
<record model="ir.ui.view" id="lista_tareas.calendar">
    <field name="name">lista_tareas calendar</field>
    <field name="model">lista_tareas.lista_tareas</field>
    <field name="arch" type="xml">
        <calendar
            color="prioridad"
            date_start="fecha_creacion"
            date_stop="fecha_terminada"
            string="Calendario de Tareas">

            <field name="tarea"/>
            <field name="prioridad"/>
            <field name="urgente"/>
            <field name="realizada"/>

        </calendar>
    </field>
</record>
```

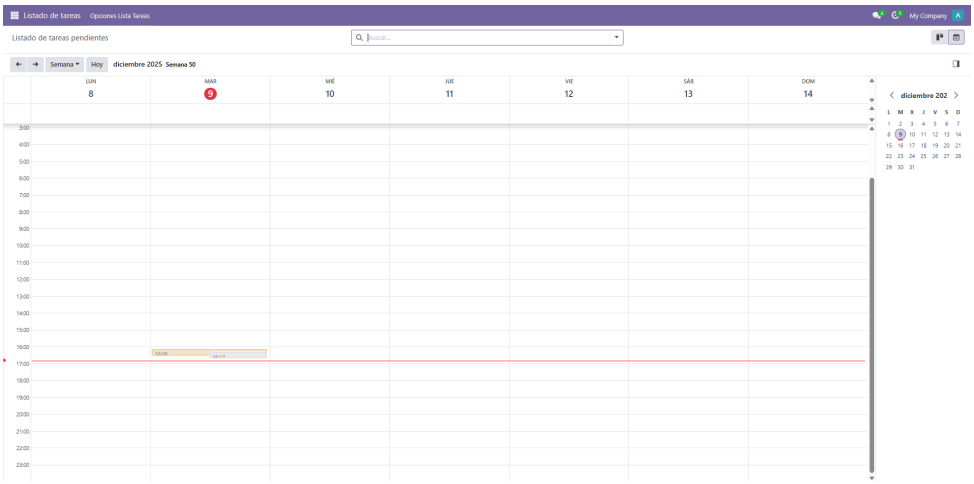
Código 2: Template para vista Calendar

Cabe recalcar que en el 'act.window' debemos añadir tanto 'kanban' como 'calendar' para que muestre las vistas correctamente y sin darnos errores, de la siguiente manera:

```xml
<record model="ir.actions.act_window" id="lista_tareas.action_window">
    <field name="name">Listado de tareas pendientes</field>
    <field name="res_model">lista_tareas.lista_tareas</field>
    <field name="view_mode">kanban,form,calendar</field>
</record>
```

Código 3: Act window con tipos de vistas

Con todo esto hecho ahora deberiamos tener la opción de cambiar a vista calendario en la parte superior derecha y se vería tal que así:


Listado de tareas
Opciones Lista Tareas
My Company
Listado de tareas pendientes
Buscar...
Semana
Hoy
diciembre 2025 Semana 50
LUN 8
MAR 9
MIÉ 10
JUE 11
VIE 12
SÁB 13
DOM 14
diciembre 202
L M X J V S D
1 2 3 4 5 6 7
8 9 10 11 12 13 14
15 16 17 18 19 20 21
22 23 24 25 26 27 28
29 30 31
3:00
4:00
5:00
6:00
7:00
8:00
9:00
10:00
11:00
12:00
13:00
14:00
15:00
16:00
17:00
18:00
19:00
20:00
21:00
22:00
23:00

2. Ampliación módulo biblioteca de cómics

En la segunda actividad vamos a modificar el módulo de biblioteca de cómics para que nos permita:

- Gestionar socios (con nombre,apellido e identificador)
- Gestionar ejemplares de préstamo (con el socio al que se ha prestado, fehca de inicio de prestamo y fecha de devolución esperada)
- Restringir fecha de préstamo (no puede ser posterior al día actual) y fecha de devolución (no puede ser anterior al día actual)

Empezaremos por la gestión de los socios. Para esto crearemos el modelo 'socios.py' y el view correspondiente. En el modelo pondremos los campos mencionados anteriormente para lo socios, indicandole el nombre, si es requerido y una ayuda (por si el nombre no es suficientemente descriptivo). También tuve que añadir un 'sql.constraints' para hacer único el identificador.

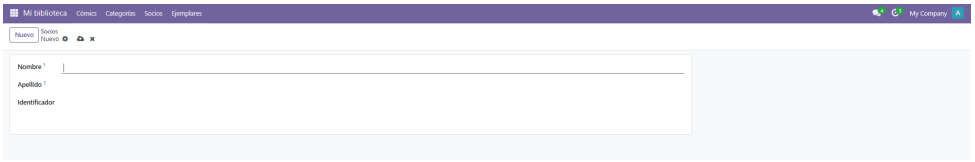
```
    nombre = fields.Char(
        string='Nombre',
        required=True,
        help='Nombre del socio'
    )

    apellido = fields.Char(
    string='Apellido',
    required=True,
    help='Apellido del socio'
    )

    identificador = fields.Text(
        string='Identificador',
        required=True,
    )

    _sql_constraints = [
        ('identificador_unique', 'unique(identificador)', 'El
           identificador debe ser unico')
    ]
```

Código 4: Modelo de los socios

Para mostrar esto, añadiremos el view del socio como 'socio.xml', en el añadiremos toda la parte visual referente a los socios, como el boton para acceder desde el menú, el formulario para crear socios o la lista donde se mostrará la información de estos.

Configuraremos la vista a nuestro gusto, siempre y cuando cumpla con lo pedido. En mi caso la vista principal quedó tal que así:

Y el formulario se ve así:



Ahora para los ejemplares crearemos un nuevo modelo llamado 'ejemplar.py'. Declararemos los campos que vamos a necesitar para el ejemplar que serán: el nombre del ejemplar, el id del comic, el id del socio, la fecha de prestamo, la fecha de devolución y el estado (disponible o no).

```
        nombre = fields.Char(
        string='Nombre',
        related='comic_id.nombre',
        readonly=True
    )
    comic_id = fields.Many2one(
        'biblioteca.comic',
        string='Comic',
        required=True
    )
    socio_id = fields.Many2one(
        'biblioteca.comic.socio',
        string='Prestado a'
    )
    fecha_prestamo = fields.Date(
        string='Fecha de prestamo'
    )
    fecha_devolucion = fields.Date(
        string='Fecha prevista de devolucion'
    )
    estado = fields.Selection(
        [('disponible', 'Disponible'), ('prestado', 'Prestado')],
        string='Estado',
        compute='_compute_estado',
```

```
        store=True
    )
```

Código 5: Modelo de los ejemplares

Después de esto vamos a controlar las fechas como se nos pide con 'api.constraints', utilizando un 'if' para comprobar si la fecha es anterior o posterior al día actual:

```
    @api.constrains('fecha_prestamo')
    def _check_fecha_prestamo(self):
        for record in self:
            if record.fecha_prestamo and record.fecha_prestamo > date.
               today():
                raise ValidationError('La fecha de prestamo no puede ser
                    posterior al dia actual')

    @api.constrains('fecha_devolucion')
    def _check_fecha_devolucion(self):
        for record in self:
            if record.fecha_devolucion and record.fecha_devolucion <
               date.today():
                raise ValidationError('La fecha prevista de devolucion
                   no puede ser anterior al dia actual')
```

Código 6: Comprobar fecha válida

Por último vamos a indicar el estado del ejemplar. Utilizaremos un campo computado teniendo como referencia el id del socio y la fecha de préstamo, en caso de tener un socio asignado (siendo al que se le ha prestado) y la fecha de préstamo indicada, se marcará como tal.

```
    @api.depends('socio_id', 'fecha_prestamo')
    def _compute_estado(self):
        for record in self:
            if record.socio_id and record.fecha_prestamo:
                record.estado = 'prestado'
            else:
                record.estado = 'disponible'
```

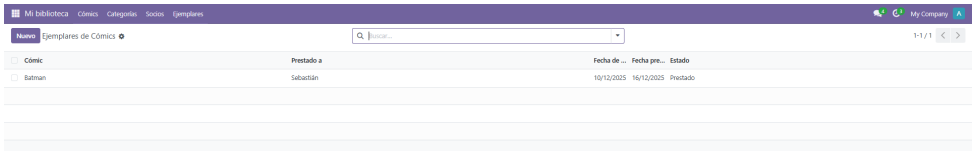
Código 7: Indicar estado del ejemplar

Antes de continuar, vamos a cambiar el .csv en la carpeta 'security' para después tener acceso y poder ver las funcionalidades creadas. En este archivo indicaremos el id, el nombre del permiso, el modelo al que se aplica, grupo de usuarios al que se aplica y los permisos que le daremos (lectura,escritura,creación,eliminación):

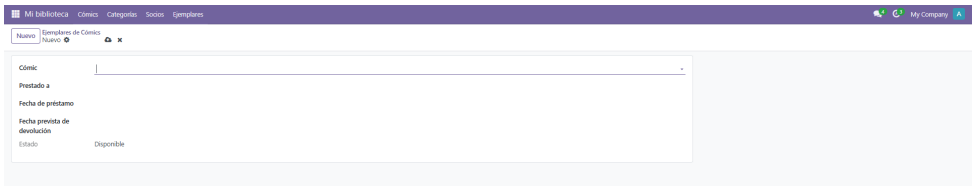
```
id,name,model_id:id,group_id:id,perm_read,perm_write,perm_create,
    perm_unlink
access_biblioteca_comic_user,biblioteca_comic_user,
    model_biblioteca_comic,base.group_user,1,1,1,1
access_biblioteca_categoria_user,biblioteca_comic_categoria_user,
    model_biblioteca_comic_categoria,base.group_user,1,1,1,1
access_biblioteca_socio_user,biblioteca_comic_socio_user,
    model_biblioteca_comic_socio,base.group_user,1,1,1,1
access_biblioteca_ejemplar_user,biblioteca_comic_ejemplar_user,
    model_biblioteca_comic_ejemplar,base.group_user,1,1,1,1
```

Código 8: CSV de seguridad

Con las funcionalidades lista podemos pasar a hacer la vista que llamaremos 'biblioteca_comic_ejemplar.xml'. Indicaremos lo de siempre, como se debe mostrar en el menú, como será el formulario para crear nuevos ejemplares y como se verá la lista de ellos. En mi caso la vista principal se ve de la siguiente manera:

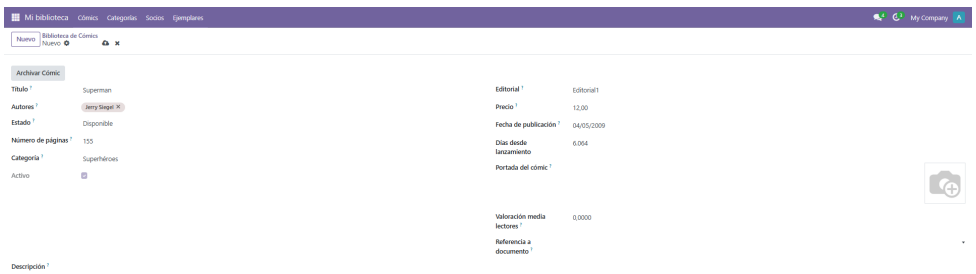


Y el formulario de creación de ejemplares es tal que así:

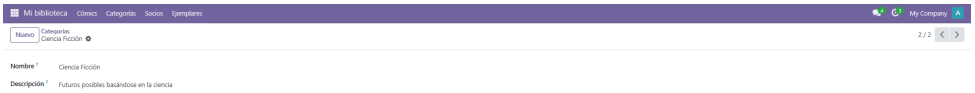


Ahora que tenemos todo hecho vamos a probar a crear un ejemplo para cada modelo y ver los diferentes campos que tenemos para rellenar.

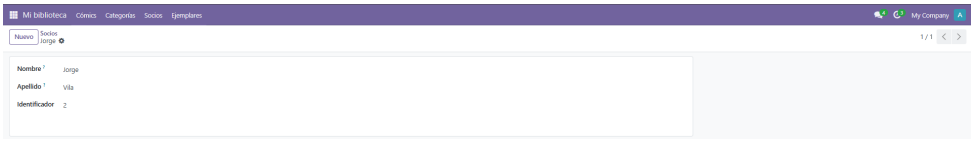
Por un lado en los cómics vamos a crear un ejemplo de Superman:



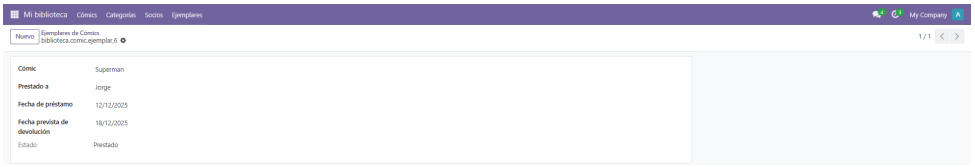
Esta sería la creación de una categoría:



Añadiremos un nuevo socio:



Y por último crearemos un ejemplar del cómic que hemos creado y se lo llevará prestado el usuario que hemos creado:



3. Creación módulo de un hospital

En esta actividad vamos a crear un módulo para un hospital en el que se tenga en cuenta lo siguiente:

- Pacientes: Tienen nombre, apellidos y síntomas
- Médicos: Tienen nombre, apellidos y número de colegiado
- Consulta: Un médico atiende a un paciente.
- Relaciones: Un paciente podrá ser atendido por varios médicos y un médico podrá atender a varios pacientes

Con esto en cuenta, empezaremos creando la base del módulo con el comando 'scaffold', primero accedemos a nuestro contenedor con odoo, ejecutamos el comando poniendo el nombre que queremos para nuestro módulo (en este caso hospital) y le indicamos la ruta donde debe crearlo, a mayores le pondremos todos los permisos con 'chmod' por si acaso:

```
    docker exec -it odoo18 bash
    odoo scaffold hospital /mnt/extra-addons
    chmod 777 -R /mnt/extra-addons/hospital
```

Código 9: Scaffold para módulo

Es importante añadir en __init__.py el import de los modelos, podemos hacerlo ahora o después de crearlos, es indiferente.

```
from . import models
from . import paciente
from . import medico
from . import consulta
```

Código 10: Imports en init.py

Con la base preparada, vamos a empezar con la creación de los modelos. Usaremos un modelo para médicos, otro para pacientes y un tercero para consultas. Empezemos por el de médicos:

En este módulo vamos a indicar los campos que tendrá, que serán el nombre (nombre y apellidos), número de colegiado y la relación de la consulta (esta relación será un 'One2Many' ya que un médico puede tener X consultas, pero cada consulta la realiza un único médico)

```
    from odoo import models, fields

    class Medico(models.Model):
        _name = 'hospital.medico'
        _description = 'Medico'

        name = fields.Char(string="Nombre y apellidos", required=True)
        numero_colegiado = fields.Char(string="Numero de colegiado",
           required=True)

        consulta_ids = fields.One2many(
            'hospital.consulta',
            'medico_id',
            string="Consultas"
        )
```

Código 11: Modelo médico

En el modelo del paciente tendremos algo muy similar, simplemente cambiaremos el campo de número de colegiado por el de síntomas que se nos pide (La relación sigue siendo igual).

```
    from odoo import models, fields

    class Paciente(models.Model):
        _name = 'hospital.paciente'
        _description = 'Paciente'

        name = fields.Char(string="Nombre y apellidos",required=True)
        sintomas = fields.Text(string="Sintomas")

        consulta_ids = fields.One2many(
            'hospital.consulta',
            'paciente_id',
            string="Consultas"
        )
```

Código 12: Modelo paciente

Por último, el modelo de la consulta tendrá 'la otra parte' de las relaciones para médicos y pacientes (siendo 'Many2one'), además de el diagnóstico y una fecha.

```
from odoo import models, fields

class Consulta(models.Model):
    _name = 'hospital.consulta'
```

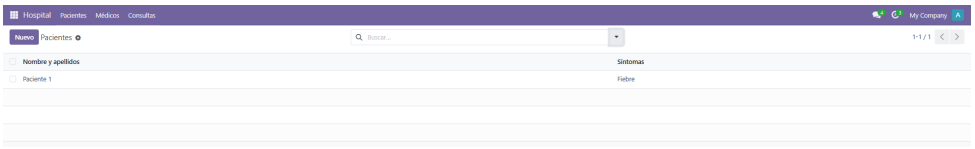
```
    _description = 'Consulta Medica'

    paciente_id = fields.Many2one(
        'hospital.paciente',
        string="Paciente",
        required=True
    )

    medico_id = fields.Many2one(
        'hospital.medico',
        string="Medico",
        required=True
    )

    diagnostico = fields.Text(string="Diagnostico")
    fecha = fields.Datetime(string="Fecha", default=fields.Datetime.now)
```

Código 13: Modelo consulta

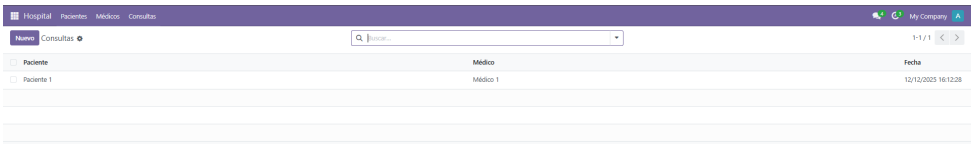
Después de terminar los modelos vamos a diseñar las vistas. En mi caso utilizaré 'list' (para mostrar los existentes) y 'form' (para la creación de nuevos). Empezemos por los médicos.

Creamos 'medico.xml', en el indicaremos lo que se mostrará en la lista y los campos para el formulario. Mi vista es la siguiente:

Después, tenemos la vista para los pacientes, que será prácticamente igual a la de los médicos:

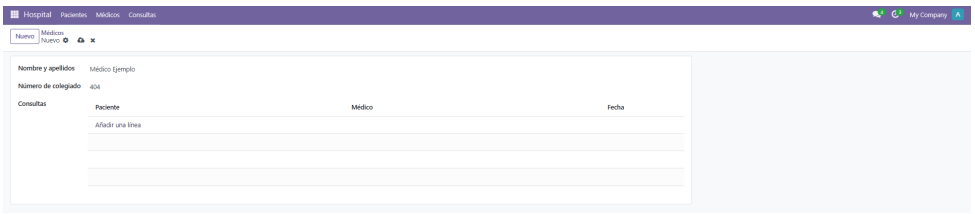


Como última vista tenemos la de las consultas, está es algo diferente ya que tiene los dos campos que toman un ID, el de médico y el de paciente, junto con una fecha.

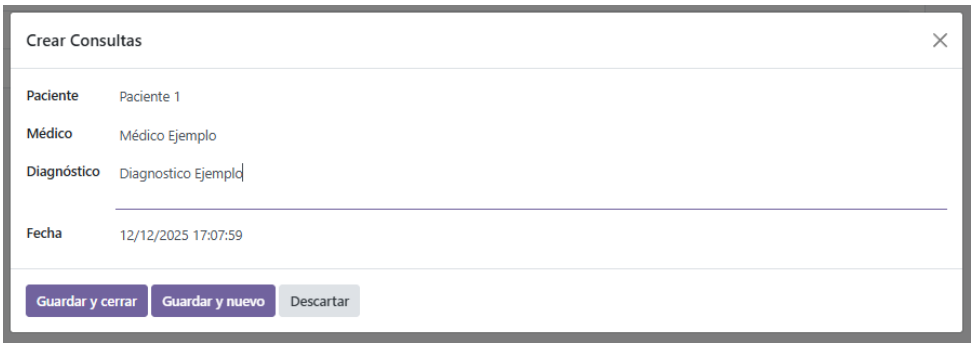


Con todo esto hecho, podemos probar a crear unos ejemplos y aprovechar para ver los formularios.

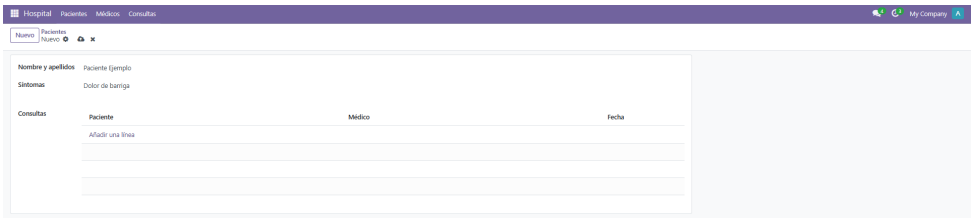
Empezaremos creando un médico, podremos indicar los campos ya mencionados anteriormente.



También en el propio formulario tenemos para añadir consultas directamente a este médico.

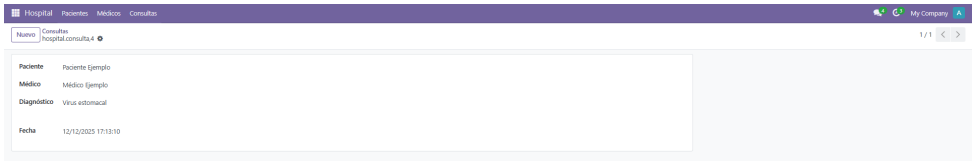


Ahora crearemos un nuevo paciente (con los campos ya mencionados anteriormente):



Y al igual que con los médicos podemos indicar una consulta directamente aquí con alguno de los médicos:

Por último crearemos una consulta con el médico y paciente que acabamos de crear y le indicaremos un diagnóstico:



4. Creación de módulo de ciclo formativo

Cuando iba a empezar este módulo tuve que eliminar la BBDD que tenía previamente en Odoo porque estaba corrupta y no podía iniciarla, asi que tuve que volver a montar el Odoo (por si hay algunos nombres diferentes)

La última actividad consistirá en la creación de un módulo que represente los estudios de ciclos formativos en un instituto. Usaremos los siguiente modelos:

- Ciclo: Representa un ciclo formativo en el instituto.
- Módulo: Se relaciona con un ciclo formativo (pertenece), los alumnos (matriculados) y los profesores (que lo imparten).
- Alumno: Representa un alumno que se matricula en módulos.
- Profesor: Imparte diferentes módulos.

Empezaremos montando el módulo base con 'scaffold' como ya explicamos en la actividad anterior y añadiremos los modelos que usemos al init__.py. Ya con la base montada nos pondremos a trabajar.

En este caso vamos a necesitar cuatro modelos (los listados anteriormente) en total. Vamos a empezar por el de los ciclos.

Para este modelo solo necesitaremos dos cosas, el nombre del ciclo y una relación con los módulos (que será 'One2many' debido a que un ciclo puede tener X módulos)

```
from odoo import models, fields

class Ciclo(models.Model):
    _name = 'instituto.ciclo'
    _description = 'Ciclos Formativos'

    name = fields.Char(string='Nombre del Ciclo',required=True)
    modulo_ids = fields.One2many('instituto.modulo','ciclo_id',string='
        Modulos')
```

Código 14: Modelo ciclo

Seguimos con los módulos, para este necesitaremos también un nombre, pero más importante todavía, vamos a referenciar a todos los modelos (tanto el ciclo que ya hemos creado como los dos que nos faltan), gestionamos las relaciones de manera que el módulo tenga X alumnos, pero solo pertenece a un ciclo y solo lo imparte un profesor.

```
from odoo import models, fields

class Modulo(models.Model):
    _name = 'instituto.modulo'
    _description = 'Modulo'

    name = fields.Char(string='Nombre del Modulo', required=True)
    ciclo_id = fields.Many2one('instituto.ciclo', string='Ciclo
       Formativo', required=True)
    alumnos_ids = fields.Many2many('instituto.alumno', string='Alumnos
       Matriculados')
    profesor_id = fields.Many2one('instituto.profesor', string='Profesor
       ')
```

Código 15: Modelo módulo

El tercer modelo será el de alumno, que similar al de los ciclos, tendrá un nombre y en este caso una referencia 'Many2many' ya que un alumno tiene X módulos y un módulo tiene X alumnos.

```
from odoo import models, fields

class Alumno(models.Model):
    _name = 'instituto.alumno'
    _description = 'Alumno'

    name = fields.Char(string='Nombre del Alumno',required=True)
    modulo_ids = fields.Many2many('instituto.modulo', string='Modulos
       del Alumno')
```

Código 16: Modelo alumno

El último modelo será el de profesor, que es prácticamente igual al de los alumnos, teniendo un nombre y una relación 'Many2many'.

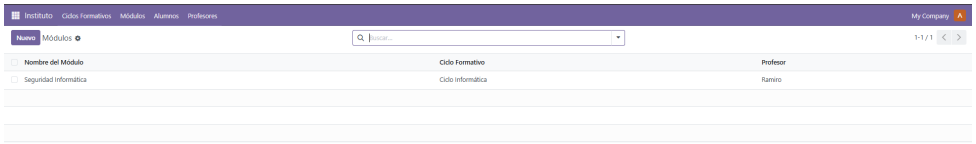
```
from odoo import models, fields

class Profesor(models.Model):
    _name = 'instituto.profesor'
    _description = 'Profesor'

    name = fields.Char(string='Nombre del Profesor',required=True)
    modulo_ids = fields.Many2many('instituto.modulo', string='Modulos
       del Profesor')
```

Código 17: Modelo profesor

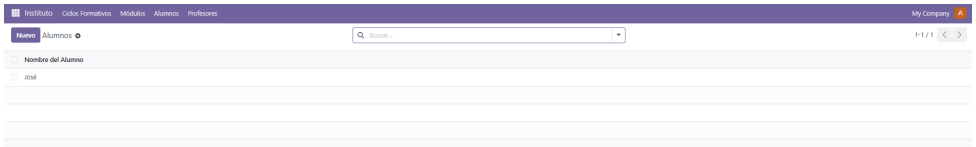
Ahora que tenemos los modelos creados, haremos las vistas para cada uno de ellos. Las vistas serán básicas, contando con una lista para ver los existentes y un formulario para la creación.

La vista de los ciclos es tal que así, mostrando el nombre:

La vista de módulos es la más completa, puedes ver el nombre del módulo, el ciclo al que pertenece y el profesor que lo imparte:



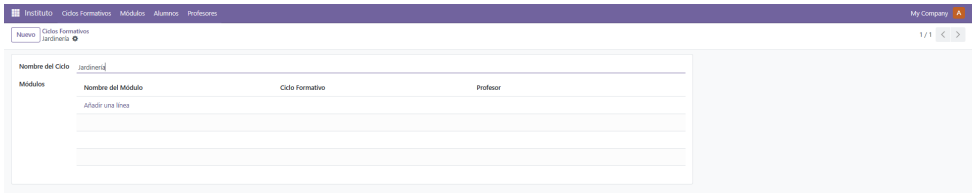
En la vista de alumnos podemos ver sus nombres:



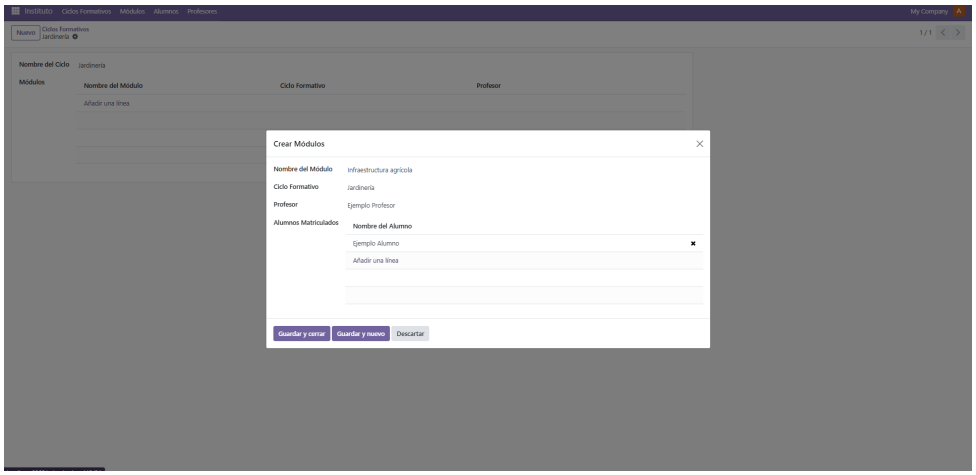
Y para los profesores también podemos ver sus nombres:

Ahora que tenemos las vistas vamos a probar la creación de cada uno de los modelos:

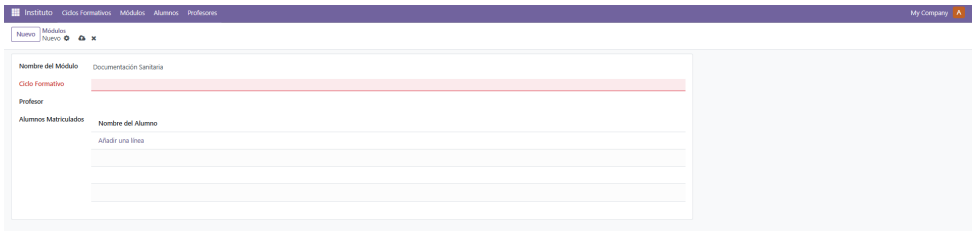
Vamos a empezar creando un ciclo, podremos ponerle un nombre y meterlo directamente en un módulo:



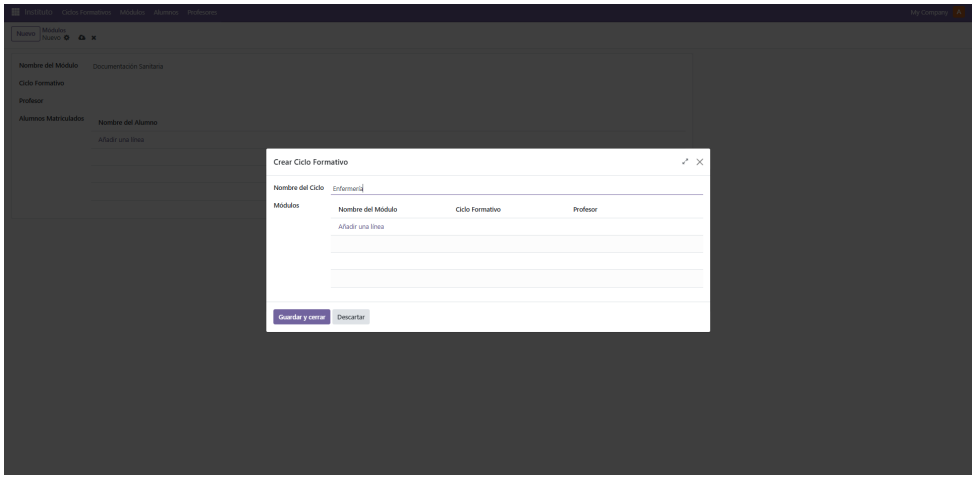
O podemos crear el módulo desde aquí mismo y añadírselo:



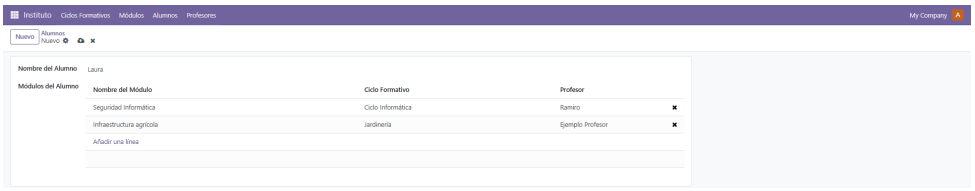
Ahora vamos a crear un módulo (que va a ser igual que la creación que acabamos de hacer desde ciclos).



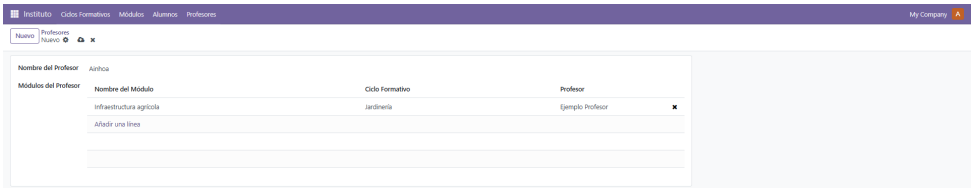
Desde aquí podremos crear todo lo necesario si es que ya no están creado previamente, como ciclos, alumnos y profesores.



Ahora en alumnos podremos ponerles su nombre, y añadirlos al módulo/s que sea/n necesarios.



Y por último crearemos un profesor, también con su nombre y los módulos que imparte:



Ahora que tenemos todos los modelos y vistas nos queda ver la configuración de seguridad. Crearemos un '.xml' en la carpeta **'security'**.

Dentro de el empezaremos definiendo los dos grupos de usuarios que queremos (profesor y director):

```
    <record id="grupo_director" model="res.groups">
        <field name="name">Director</field>
    </record>

    <record id="grupo_profesor" model="res.groups">
        <field name="name">Profesor</field>
    </record>
```

Código 18: Grupos de usuarios

Ahora definiremos los permisos propiamente dichos, línea por línea iremos indicando una regla y el modelo al que se le va a aplicar, luego definimos el 'filtro' que queremos (basicamente permitir a los profesores ver a sus iguales) y limitamos esa regla al grupo profesor. Para finalizar indicamos los permisos que tendrán (en este caso, solo lectura).

```
    <record id="profesor_regla" model="ir.rule">
        <field name="name">Profesor solo lectura de profesores</field>
        <field name="model_id" ref="model_instituto_profesor"/>
        <field name="domain_force">[(1,'=',1)]</field>
        <field name="groups" eval="[(4, ref('ciclos.grupo_profesor'))
           ]"/>
        <field name="perm_read" eval="True"/>
        <field name="perm_write" eval="False"/>
        <field name="perm_create" eval="False"/>
```

```
9           <field name="perm_unlink" eval="False"/>
10      </record>
```

Código 19: Regla de seguridad para profesor

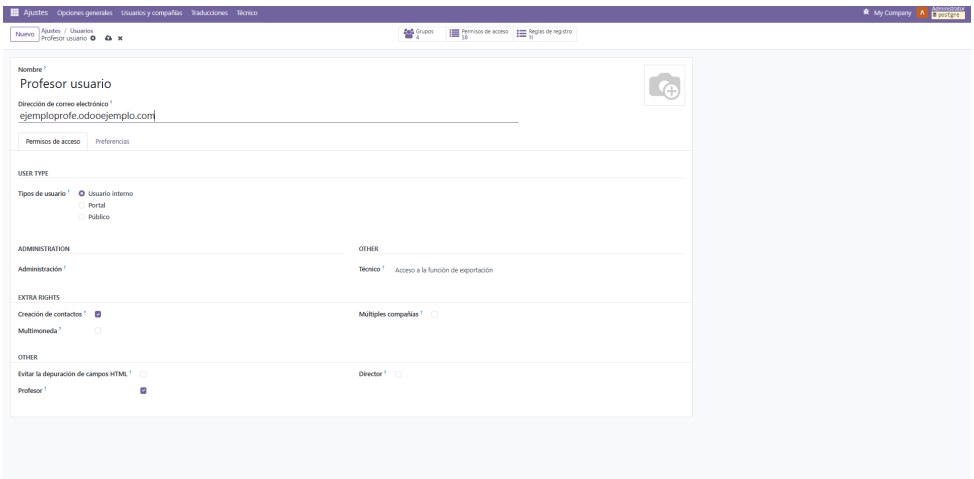
Para los directores evitamos añadir otra regla ya que la el predefinido para Odoo será lo que indiquemos en nuestro '.csv' que está en la carpeta 'security:

```
1 id,name,model_id:id,group_id:id,perm_read,perm_write,perm_create,
    perm_unlink
2 access_alumno,access_alumno,model_instituto_alumno,,1,1,1,1
3 access_modulo,access_modulo,model_instituto_modulo,,1,1,1,1
4 access_profesor_read,access_profesor_read,model_instituto_profesor,
    grupo_profesor,1,0,0,0
5 access_profesor,access_profesor,model_instituto_profesor,grupo_director
    ,1,1,1,1
6 access_ciclo,access_ciclo,model_instituto_ciclo,,1,1,1,1
```

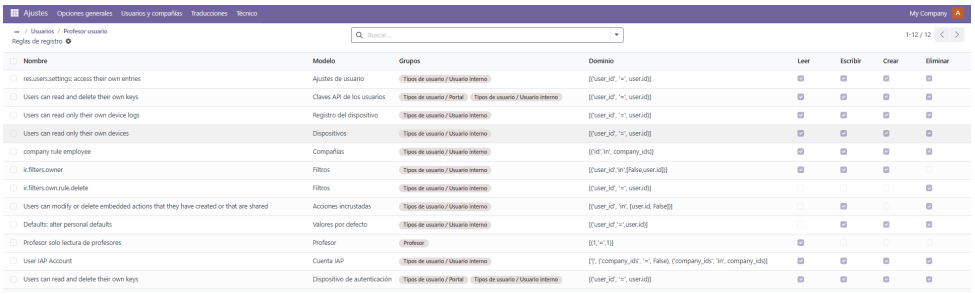
Código 20: CSV con permisos

Esto quiere decir que todos los usuarios (en nuestro caso solo tenemos profesor, director y administrador), tendrán todos los permisos, pero con lo anterior en el '.xml' limitamos solo a los profesores

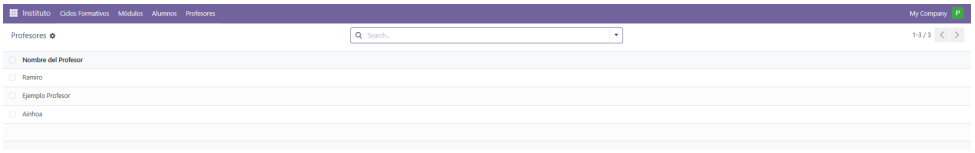
Podemos verlo de manera más visual si creamos un usuario profesor marcando la casilla de 'profesor':



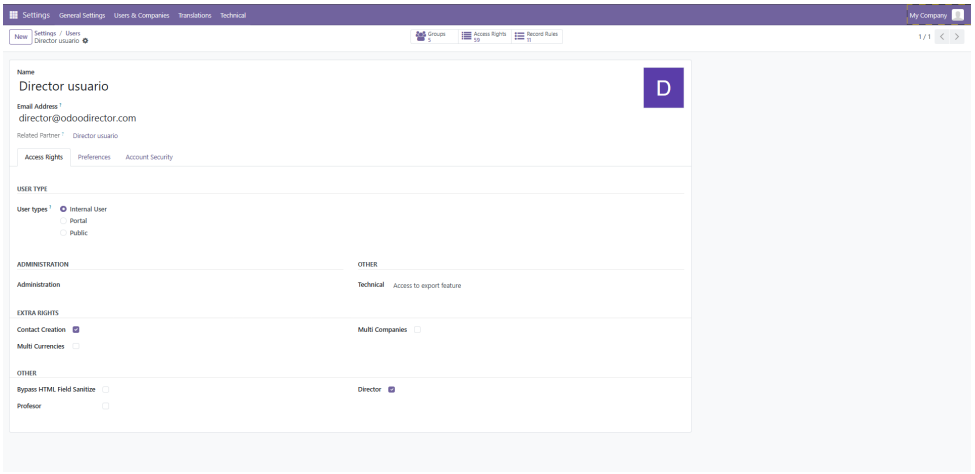
Podemos acceder a sus reglas de registro y aquí encontraremos nuestra regla 'Profesor solo lectura de profesores' para limitarlo a solo lectura:



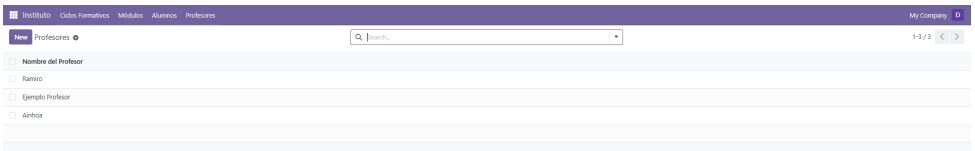
Y si vamos a profesores con este usuario vemos que solo podemos leer los profesor actuales, pero no modificar, crear, o borrar nada.



Ahora probaremos creando un director marcando la casilla correspondiente:



En este caso la regla no aplica para los directores así que no aparecera, pero podemos comprobar que tienen todos los permisos viendo la pestaña de profesores:



Con la actividad 4 terminada, queda finalizada la práctica de creación de módulos y vistas en Odoo