Practica 4

Módulos Odoo Modelo y vista

1. Introducción. Estructura inicial.
2. Actividad 01
    a. Vista de las tareas en formato KANBAN.
    b. Vista de las tareas con formato Calendar.
3. Actividad 02
    a. Gestionar Socios
    b. Ejemplares para préstamos.
    c. Restricciones.
4. Actividad 03
    a. Estructuración de carpetas
    b. Creación de modelos
    c. Creación de vistas
    d. Configurar Manifest
5. Actividad 04
    a. Modelo ciclo Formativo
    b. Modelo módulo
    c. Modelo Alumno
    d. Modelo Profesor
    e. Demostración

Actividad 01

Vista Kanban

Lo primero que vamos a hacer será crear un nuevo campo de estados, para ello:

-Definimos el nuevo campo de estados en el archivo models.py

```
estado = fields.Selection(
    [
        ('todo', 'Por hacer'),
        ('doing', 'Haciendo'),
        ('done', 'Hecha'),
    ],
    string="Estado",
    default='todo',
)
```

Me he basado en el típico ejemplo de "KANBAN TO DO" con 3 estados.

Los estados son

- POR HACER : 'todo'
- HACIENDO : 'doing'
- HECHO : 'done'

Luego, en el views.xml:

-Añadimos el campo de :

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

Tanto en la vista de lista como en la vista KANBAN

Dentro del Template de kanban, debemos añadir el campo de estado:

```
<t t-name="kanban-box">

<div>Estado: <field name="estado"/></div>
```

Esto hará que visualicemos el dato en la carta.

Luego, especificaremos en el contenedor de acción, que agrupe según el campo de "Estado":

```
1    <field name="context">{'group_by': 'estado'}</field>
```

A mayores, le he puesto que por defecto (sino recupera el dato, que no se porque no iba a poder recuperarlo) ponga "todo", codigo de 'Por hacer'.

{img/kanba.png}

Este es el resultado, se va actualizando a medida que vamos cambiando los estados de las tareas.

Vista Calendario

Vamos al archivo models.py y añadimos la variable fecha:

```
fecha = fields.Date(string="Fecha")
```

Aclaramos que será un campo de tipo date.

Luego, accedemos al archivo de views.xml

Una vez estemos dentro del fichero, añadimos al igual que hicimos con el campo de estado. El campo de fecha:

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

También he añadido el campo en la vista de Kanban, entiendo que no es obligatorio pero asi todas las vistas son iguales.

Lógicamente, como hicimos con la vista de kanban, tendremos que crear una nueva vista para que se mire en formato calendario.

Esta vista será la vista .calendar, mostraremos cada tarea en formato calendario a partir de un campo date.

Para ello, añadí el siguiente bloque dentro de las vistas (data)

```
<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 date_start="fecha"
string="Calendario de tareas">

            <field name="tarea"/>

        </calendar>

    </field>

</record>
```

Una vez creada la vista, debemos aclarar el view_mode con el formato correcto - CALENDAR

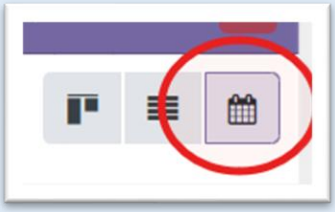
```
<field name="view_mode">kanban,tree,calendar,form</field>
```

Actualizar web. Como siempre, tras realizar los cambios, lanzamos por la cmd:

```
docker compose restart web
```

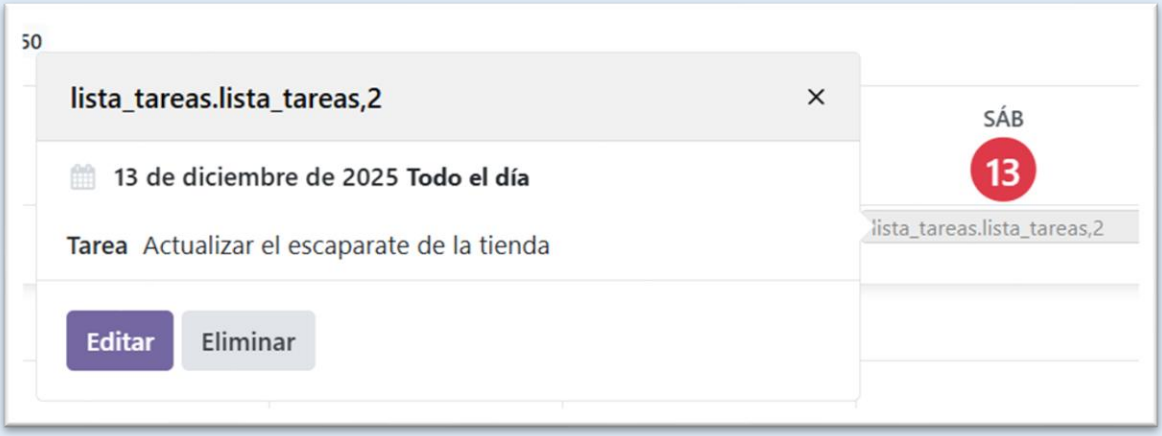
Entramos dentro del odoo y hacemos click en Actualizar lista de aplicaciones para forzar el actualzado del módulo.

Hago cntrl+f5 con el navegador abierto, y ya podemos ver que se nos ha cargado una nueva vista:



Esta vista se pone con el icono de calendario.

Ahora, ya podremos ver las tareas seccionadas por el campo de fecha. Yo le he puesto la fecha de “hoy” a una tarea y así es como la he visualizado:



Con esto, habría terminado la actividad 1.

Conclusión de pasos importantes para nuevas vistas:

- Tenemos que asegurarnos de que los campos/variables existan en el models.py, sino existen, los daremos de alta.

- Creamos la nueva vista de XML, para ellom añadimos la etiqueta:

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

  Le tendremos que añadir el id en la propia etiqueta, dentro pondremos los campos que usará la vista. Es importante que definamos dentro del campo externo el tipo de vista (como hicimos con Kanban o calendar)
- Conectamos una acción. Aquí definimos el comportamiento de la vista. Tenemos que especificar lo siguiente:

  ```
  <recordmodel="ir.actions.act_window"id="id_ejemplo">
  ```

  Aquí definimos cosas como por que debe agruparse:

```
<field name="context">{'group_by':
'estado'}</field>
```

Actividad 02

Para esta actividad he clonado el repositorio de github donde está la carpeta de “EJ03 COMICS SIMPLE”. Lo he metido dentro de la carpeta de addons, no he tenido que realizar ninguna configuración extra mas que comentar esta línea del archivo __init__.py:

```
#from . import controllers
```

Ya que yo no tengo ninguna carpeta de controllers en mi repositorio de odoo.

Gestionar socios.

Debemos permitir gestionar en el módulo que acabamos de añadir, socios almacenando su nombre, apellido e identificador.

Para ello, lo primero que debemos realizar es crear el modelo de socios dentro de la carpetra de models. He generado un archivo *socio.py*.

Dentro, he definido lo que tiene que tomar de la raíz del repositorio y la clase llamada “BibliotecaSocio” (al igual que la clase de ejempo de biblioteca_comic). Tiene el siguiente formato:

```python
class BibliotecaSocio(models.Model):
    _name = 'biblioteca.socio'
    _description = 'Socio'

    identificador = fields.Char(string="Identificador", required=True)
    nombre = fields.Char(string="Nombre", required=True)
    apellido = fields.Char(string="Apellido", required=True)
```

Definimos:

- _name: Nombre técnico del modelo (se usa para llamadas)
- Descripción.
- Fields.Char: Campos de texto que acompañan la clase. Como en la actividad se pide, todos son obligatorios. Lo defino con: *required*=True

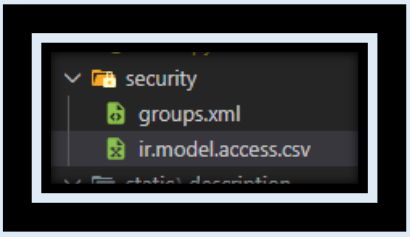
Ahora, tendremos que hacer que odoo cargue este archivo Python. Pensaba que esto se hacía automáticamente al actualizar el módulo, pero no. Tendremos que definirlo dentro del archivo __init__.py dentro de la carpeta de models.

He añadido la siguiente línea:

```
from . import socio
```

A simple vista parecería que esta todo listo para que nuestro módulo cargue la clase de socio. Pero, aun que estructuralmente no tengamos que hacer nada más, será necesario darle permisos.

Para ello, abrimos la carpeta security. Dentro, encontramos el siguiente archivo:



Será necesario que especifiquemos que permisos tiene:

```
access_biblioteca_socio,bilbioteca.socio,model_bibl
ioteca_socio,base.group_user,1,1,1,1
```

Cada 1 representa un afirmativo.

Ahora, aun que ya tenemos la suficiente estructura para crear socios, tendremos que hacer algo igual de importante, crear una vista para visualizar y crear los socios.

Como hemos hecho en la actividad anterior, tendremos que ir al archivo views.xml y añadir dentro una vista lista.

He añadido la vista de esta manera:

```xml
<record id="biblioteca_socio_tree" model="ir.ui.view">
    <field name="name">biblioteca.socio.tree</field>
    <field name="model">biblioteca.socio</field>
    <field name="arch" type="xml">
        <tree>
            <field name="identificador"/>
            <field name="nombre"/>
            <field name="apellido"/>
        </tree>
    </field>
</record>

```

Y, como en la anterior actividad (ya creado) un formulario de creación al que le he dado el siguiente formato de formulario:

```xml
<record id="biblioteca_socio_form" model="ir.ui.view">
    <field name="name">biblioteca.socio.form</field>
    <field name="model">biblioteca.socio</field>
    <field name="arch" type="xml">
        <form string="Socio">
            <group>
                <field name="identificador"/>
                <field name="nombre"/>
                <field name="apellido"/>
            </group>
        </form>
    </field>
</record>
```

Ahora sí, el módulo esta listo para desplegarse, por lo que, lanzamos por terminal nuestro comando de confianza para forzar el actualizado de la web:

```
docker compose restart web
```

Y en odoo: Actualizar listado de aplicaciones.

Con esto pensaba que ya estaba todo listo para ser usado, pero cuando entraba en el modulo, no me salía ninguna opción para visualizar “Socios” . Si que me salía arriba la opción de comics pero no me salía nada de socios. Resultaba que me quedaba hacer algo importante. Un menuitem para activar la vista. Ya tenía creado el de comics con este formato:

```
<menuitem name="Comics" id="biblioteca_comic_menu"
parent="biblioteca_base_menu"
action="biblioteca_comic_action"/>
```

Por lo que cree el de socios aptovechando el mismo formato con los id correcto:

```
1  <menuitem id="biblioteca_socio_menu"name=
   "Socios"parent="biblioteca_base_menu"action=
   "biblioteca_socio_action"/>
```

Igualmente, pensaba que ya estaba todo, pero no. Faltaba crear una action que active la vista. Para ello, cogí la del ejemplo del comic y le puse el id del de la etiqueta *menuitem* que mostré antes.

Lo deje con el siguiente formato:

```
1  <menuitem id="biblioteca_socio_menu"name=
   "Socios"parent="biblioteca_base_menu"action=
   "biblioteca_socio_action"/>
```

Ahora sí, actualizamos la lista de aplicaciones y…

Me sigue sin ir. No se actualiza el módulo. No logro ver los cambios en el mismo por mas que fuerce actualizados. Tras mas de 1h y 15 minutos, he logrado encontrar lo que provocaba que no se actualizase.

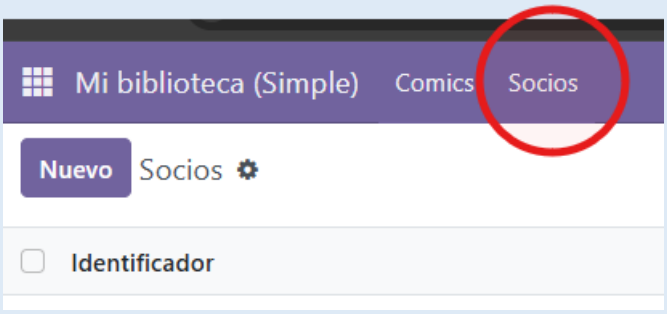
En mi biblioteca_comic.xml, puse de esta manera el menuitem:

```
<menuitem
id="biblioteca_socio_menu"name="Socios"parent="biblioteca_ba
se_menu"action="biblioteca_socio_action"/>
```

Si te fijas, no hay espacios entre cada elemento. Después de declarar el id “” empieza el name y así con todo. Por increíble que parezca. Esto ocasionaba el error. He separado los elementos con un simple “ ” y ahora,FUNCIONA:

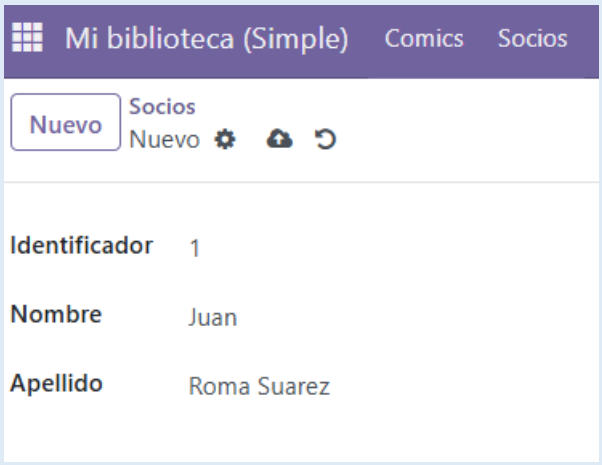
```
<menuitem id="biblioteca_socio_menu" name="Socios"
parent="biblioteca_base_menu"
action="biblioteca_socio_action"/>
```

Actualizamos la lista de aplicaciones y…

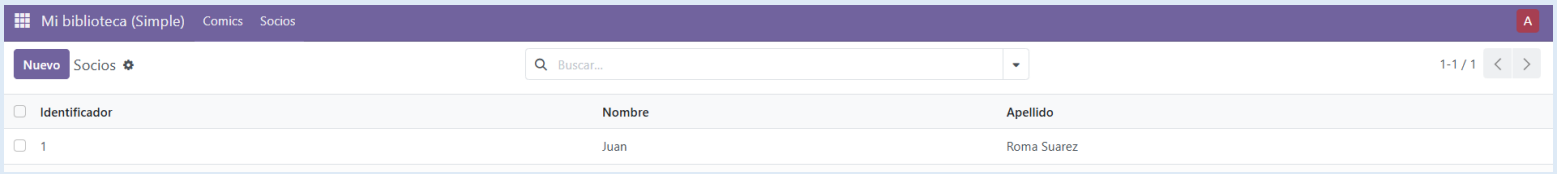


Funciona, tenemos indexado el socios.

Desde aquí , podremos dar socios de alta con el formulario:



Y poder visualizarlos en lista de esta manera:



Ahora sí, doy por finalizada la primera parte de este ejercicio. Definitivamente, si estuviese trabajando para un cliente, no podría justificar el pago de esa hora…

Ejemplares para prestamos.

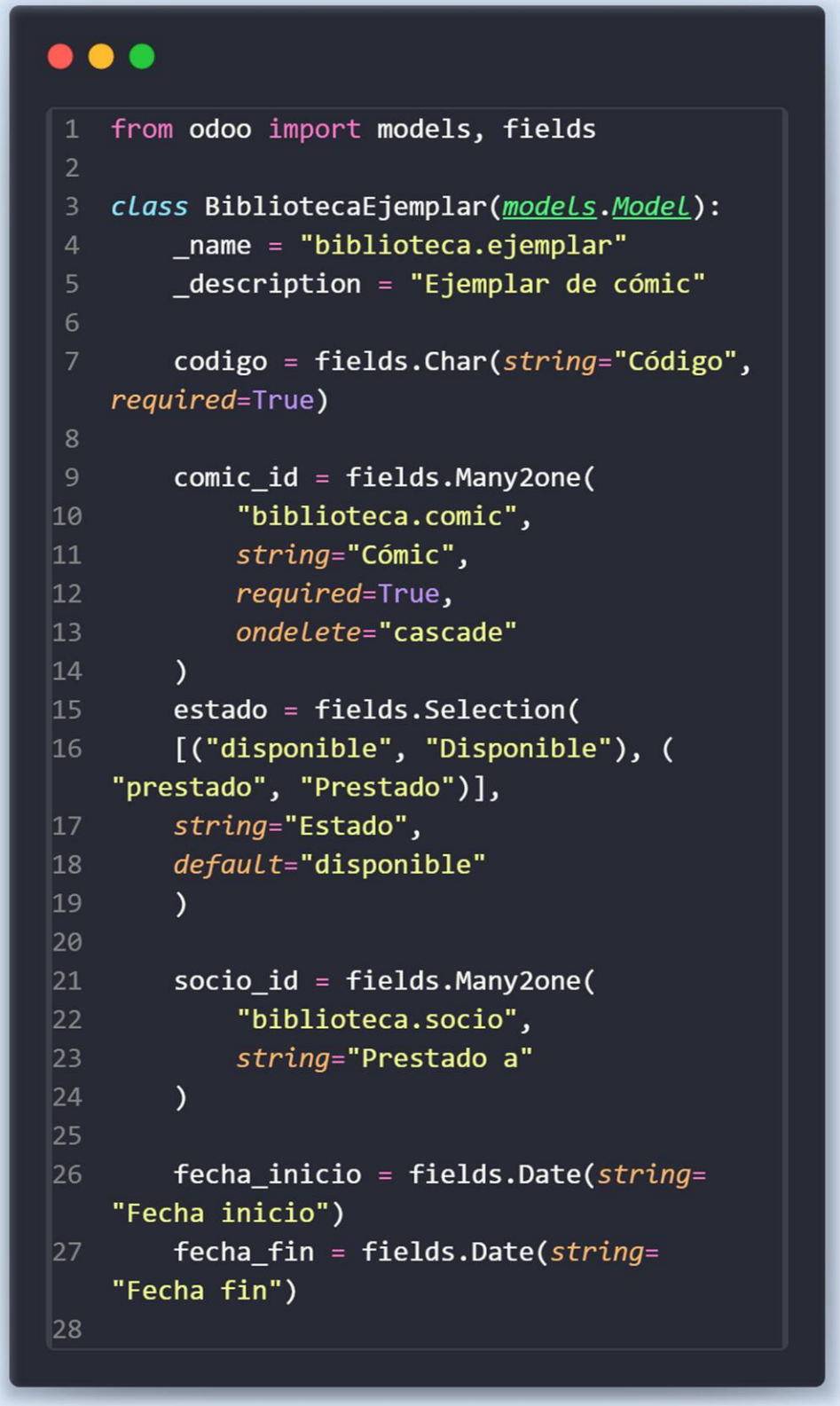
En esta sección, explicare como voy haciendo el ejercicio 2.2 y 2.3 que tratan, en resumidas cuentas de crear una funcionalidad de préstamo para comics de ejemplares. Cada ejemplar, guardará a que socio está relacionado y las fechas de fin e inicio del préstamo.

Definir los datos. He pensado en crear un nuevo modelo que sea *biblioteca.ejemplar* donde definiré:

- Comic_id – A que comic pertenece
- Código – identificador del ejemplar
- Socio_id – A que socio esta prestado ahora
- Fecha_inicio
- Fehca_fin
- Estado – Algo para saber si esta disponible o no para prestárselo a alguien.

Una vez definido al plan de ruta, nos vamos a la carpeta de models . He creado el archivo con el siguiente nombre: *ejemplar.py*

De la misma manera que antes, he definido la clase con el siguiente formato:



```python
from odoo import models, fields

class BibliotecaEjemplar(models.Model):
    _name = "biblioteca.ejemplar"
    _description = "Ejemplar de cómic"

    codigo = fields.Char(string="Código", required=True)

    comic_id = fields.Many2one(
        "biblioteca.comic",
        string="Cómic",
        required=True,
        ondelete="cascade"
    )
    estado = fields.Selection(
    [("disponible", "Disponible"), ("prestado", "Prestado")],
    string="Estado",
    default="disponible"
    )

    socio_id = fields.Many2one(
        "biblioteca.socio",
        string="Prestado a"
    )

    fecha_inicio = fields.Date(string="Fecha inicio")
    fecha_fin = fields.Date(string="Fecha fin")

```

No creo que haya mucho que explicar que no haya explicado antes. Los campos de “string” actúan como en html un label para los input de los mismos.

Busque como limitar que se repita un carácter y usé la etiqeuta de Many2one que es como una relacional en SQL. Declara, en este caso, que solo un ejemplar apunta a un socio. Es importante esto para el siguiente punto.

Importante, después, debemos decirle a odoo que debe cargar este fichero, añadiendo a la línea de __init__.py lo siguiente:

```
from . import ejemplar
```

Una vez hecho esto, será el momento de configurar enlazar los identificadores con las anteriores clases. Por lo que vamos a la clase de comic y añadimos:

```
1    ejemplar_ids = fields.One2many(
2    comodel_name="biblioteca.ejemplar",
3    inverse_name="comic_id",
4    string="Ejemplares",
5    )
```

Luego, nos vamos a la clase de socio y añadimos lo mismo modificando los id.

De esta manera, tendrmeos los comics y los socios enlazados con ejemplares.

Ahora, para que no nos pase como antes. Tendremos que darle permisos de lectura y escritura a la clase, por lo que nos vamos al archivo ir.model.access.csv y añadimos la siguiente línea:

```
access_biblioteca_ejemplar,biblioteca.ejemplar,model_biblioteca_ejemplar,base.group_user,1,1,1,1
```

Para este modelo, he considerado mejor crear una vista nueva. He entrado en la carpeta de views y he creado el archivo de ejemplar.xml. No hay nada nuevo.

Dentro, el archivo tiene:

- Vista lista:

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

  - Defino dentro las columnas que se muestran y que variable representan :

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

- Añado una vista formulario para insertar datos:

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

  - Dentro defino los campos form en dos grupos:

```
<form string="Ejemplar">
<group>
<field name="codigo"/>
<field name="comic_id"/>
<field name="estado" readonly="1"/>
</group>
<group string="Préstamo actual">
<field name="socio_id"/>
<field name="fecha_inicio"/>
<field name="fecha_fin"/>
</group>
</form>
```

- Creo el menuitem, exactamente igual que en el ejercicio anterior y la action que lo ejecuta.

Ahora, dentro del archivo __manifest__.py declaro la nueva vista:

```
'views/ejemplar.xml',
```

Ahora, en principio, sería suficiente con reiniciara el contenedor web y actualizar la lista de aplicaciones desde odoo. Lo hago y;

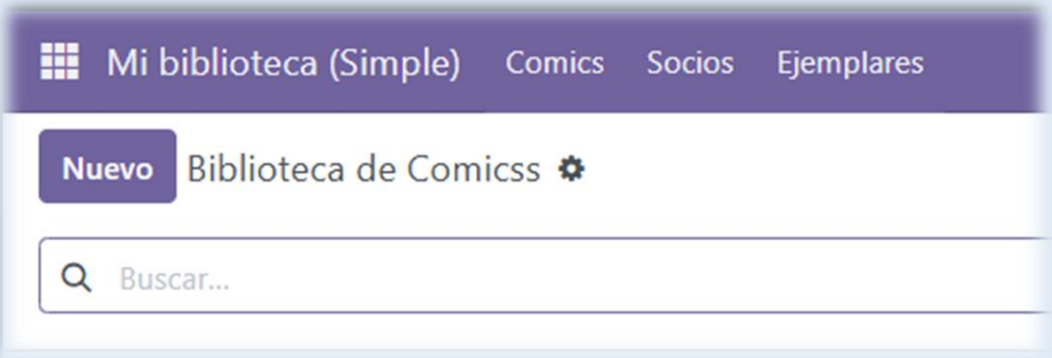
No me sale, no veo indexado la sección de “Ejemplares” cuando estoy en al vista por defecto.

Tras un buen rato revisando todo , recurrí a la IA para que revise todo mi código en búsqueda de fallos de sintaxis (como el fallo de antes) a lo que encontré lo siguiente:

```
'views/ejemplar.xml',
        'views/biblioteca_comic.xml'
```

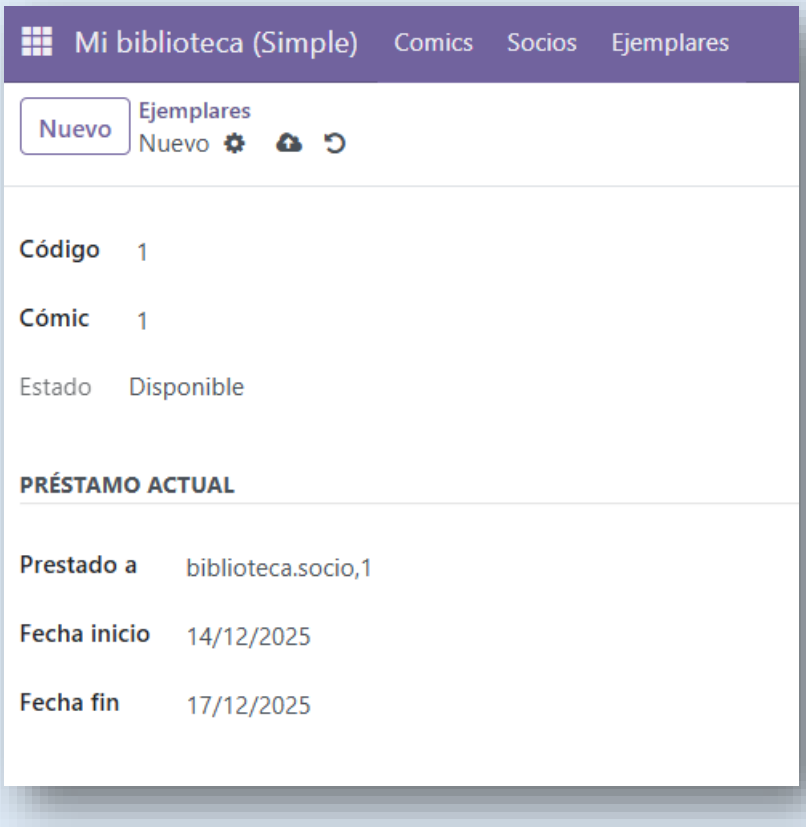
A simple vista, todo esta OK. Es posiblemente el paso mas simple, añadir la vista al manifest. Pues no, como la vista de ejemplar hereda de la de biblioteca_comic.xml, no puede estar declarada antes. Tiene que estar declarada después. Por lo que solucioné este fallo y…

Funcionó:



Ya podía acceder a la nueva vista.

Ya me dejaba completar todos los campos para registrar ejemplares de prestamos:



También, podemos visualzarlo con el formato lista:

Perfecto, pasamos al siguiente ejercicio.

Restricciones.

Esta parte del ejercicio nos pide aplicar las siguientes restricciones a lo que acabamos de hacer:

- La fecha de préstamo no puede ser posterior al día actual.
- La fecha prevista de devolución no puede ser anterior al día actual.

Para ello, debemos ir a nuestro archivo donde declaramos el modelo de ejemplar (ejemplar.py) y añadir lo siguiente.

Debemos poner api en el import ya que es algo que usaremos para hacer las restricciones:

```
from odoo import models, fields, api
```

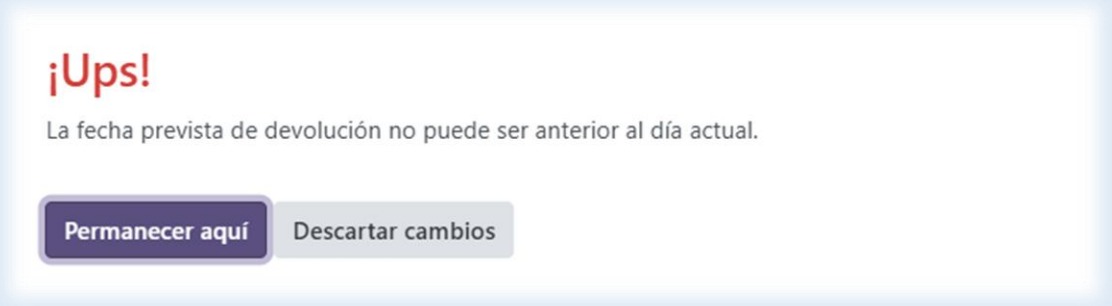
Código añadido debajo de la declaración de fechas

```
    @api.constrains("fecha_inicio", "fecha_fin")
    def _check_fechas_prestamo(self):
        hoy = fields.Date.context_today(self)

        for rec in self:
            if rec.fecha_inicio and rec.fecha_inicio > hoy:
                raise ValidationError(
"La fecha de préstamo no puede ser posterior al día actual."
)

            if rec.fecha_fin and rec.fecha_fin < hoy:
                raise ValidationError(
"La fecha prevista de devolución no puede ser anterior al día actual."
)
```

¿Cómo funciona?

- @api.contrains(variables)
  Define restricción a nivel de modelo. Odoo ejecuta el método cada vez que se da de alta una ficha de este modelo. Por lo que le decimos “cada vez que se genere, ejecuta”
- ValidationError.
  Si no se pudo crear, ¿Por qué no se pudo crear? Expulsa el resultado de porque no se pudo crear.
  - Esto me dio error. La sintaxis no funciona sino importamos ValidationError, cada vez que te falle te da “NameError:ValidationError is not defined” por lo que no puedes expulsar el mensaje de error / excepción.
- Variable con la fecha de HOY.
- Si la fecha de inicio es superior a la fecha de hoy. Expulsa el error
- Si la fecha de fin es anterior a la de hoy. Expulsa el error.

Tras reiniciar el módulo y actualizar las aplicaciones desde la interfaz de odoo. Comprobamos que la validación de errores funcione correctamente:



Tras la validación de errores, he decidido pasar a la siguiente actividad.

Aclaro, antes de comenzar, para hacer las imágenes de código, estoy usando la extensión de Visual Studio Code de CodeSnapshot. Esta librería, cuando el código es muy largo, rompe el formato del mismo, debes fijarte en la numeración de línea de la izquierda para la correcta lectura.

Actividad 3

Para esta actividad, tenemos que crear un modulo desde cero para gestionar un hospital (a baja escala). Necesitamos crear 3 modelos:

- Paciente
  - Nombre
  - Apellidos
  - Síntomas
- Medico
  - Nombre
  - Apellidos
  - Número de colegiado
- Consulta
  - Cada vez que un médico atiende a un paciente (imagino que usaré los ID de cada uno)

    - Diagnóstico

Con ello, tendremos que crear las relaciones:

- Un paciente puede haber sido atendido por varios médicos
- Un médico puede haber atendido a varios pacientes.

Estructura inicial

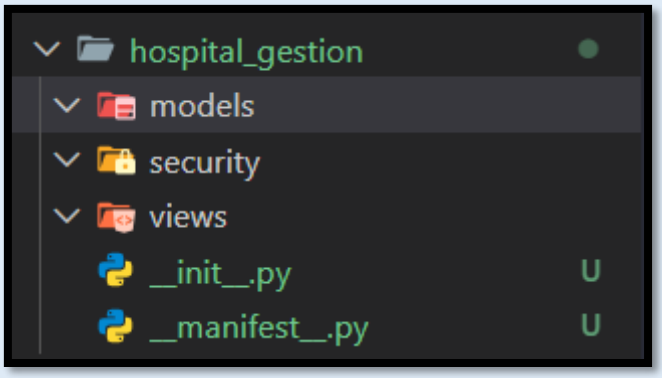
El primer paso que vamos a realizar, será crear las carpetas y archivos necesarios para el correcto funcionamiento del módulo.

Dentro de la carpeta de addons, crearemos la siguiente carpeta. *Hospital_gestion/*.

Dentro, creamos:

- __init__.py
- __manifest__.py
- Carpeta de modelos: models/
    - Dentro, __init__.py
- Views/
- Security/
    - Dentro el archivo: ir.model.access.csv

Nos debería haber quedado algo tal que así:



Ahora, ya estaríamos listos para empezar. Haré referencia a estos directorios.

Creación de modelos

Para crear modelos, debemos acceder a la carpeta de models.

Modelo de paciente.

He creado el archivo de paciente.py.

Como hablamos, es necesario guardar:

- Nombre
- Apellidos
- Sintomas

He decidido mantener el formato de los nombres de la actividad anterior. He definido la clase como HospitalPaciente. Dentro, he declarado lo siguiente:

```
class HospitalPaciente(models.Model):
    _name = "hospital.paciente"
    _description = "Paciente"

    nombre = fields.Char(string="Nombre", required=True)
    apellidos = fields.Char(string="Apellidos", required=True)
    sintomas = fields.Text(string="Síntomas")
```

Nada que no se haya dicho nunca en este documento. Campos obligatorios (required) son Nombre y Apellidos.

Modelo Médico

Para el siguiente modelo, debemos guardar:

- Nombre
- Apellidos
- Num_colegiado

He puesto todo required. No hay nada nuevo.

```python
class HospitalMedico(models.Model):
    _name = "hospital.medico"
    _description = "Médico"

    nombre = fields.Char(string="Nombre", required=True)
    apellidos = fields.Char(string="Apellidos", required=True)
    num_colegiado = fields.Char(string="Número de colegiado", required=True)

```

Modelo Consulta.

Este es quizás el mas complicado de pensar como estructurarlo. He considerado que simplemente debería guardar la relación del paciente con el médico más el diagnóstico.

Por lo que la clase debería guardar:

- Identificador de paciente
- Identificador de médico
- Diagnostico (string libre)

Debemos permitir que:

- Un paciente pueda tener muchas consultas con médicos diferentes
- Un médico puede tener muchas consultas con pacientes diferentes

A raíz de eso, he creado el archivo de consulta.py

Dentro, he definido una clase con el nombre de HospitalConsulta la cual tiene el siguiente formato:

```python
from odoo import models, fields

class HospitalConsulta(models.Model):
    _name = "hospital.consulta"
    _description = "Consulta"

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

    medico_id = fields.Many2one(
        "hospital.medico",
        string="Médico",
        required=True,
        ondelete="cascade"
    )

    diagnostico = fields.Text(string="Diagnóstico")
```

Algo expliqué antes pero, para la relación de pacientes y médicos, he declarado que sea de tipo *Many2one* Esto nos permite mantener que muchos registros pueden apuntar a un mismo.

De esta manera:

- Un mismo paciente puede tener 10 consultas
- Un médico puede tener muchas consultas

    - Cada consulta tiene un único médico

Existen las siguientes relaciones

- One2many 1:N
- Many2many N:M

Creación de vistas

Una vez configurados y creados todos los modelos, el siguiente paso que debemos realizar será el de crear las vistas donde vamos a visualizar y dar de alta estos modelos. Para ello, dentro de la carpeta de views/ he creado una vista para cada modelo:

- Paciente_views.xml
- Medico_views.xml
- Consulta_views.xml

Dentro de cada vista, he desarrollado un formulario y un modo vista para visualizar los datos. Esto está muy explicado ya en puntos anteriores. El único views donde voy a destacar algo será en la de consulta:
Paciente:

```xml
<?xml version="1.0" encoding="utf-8"?>
<odoo>

    <record id="hospital_paciente_view_tree" model="ir.ui.view">
        <field name="name">hospital.paciente.tree</field>
        <field name="model">hospital.paciente</field>
        <field name="arch" type="xml">
            <tree>
                <field name="nombre"/>
                <field name="apellidos"/>
            </tree>
        </field>
    </record>

    <record id="hospital_paciente_view_form" model="ir.ui.view">
        <field name="name">hospital.paciente.form</field>
        <field name="model">hospital.paciente</field>
        <field name="arch" type="xml">
            <form string="Paciente">
                <group>
                    <field name="nombre"/>
                    <field name="apellidos"/>
                    <field name="sintomas"/>
                </group>
            </form>
        </field>
    </record>

    <record id="hospital_paciente_action" model="ir.actions.act_window">
        <field name="name">Pacientes</field>
        <field name="res_model">hospital.paciente</field>
        <field name="view_mode">tree,form</field>
    </record>

</odoo>

```

Médico (muy réplica del anterior):

```xml
<?xml version="1.0" encoding="utf-8"?>
<odoo>

    <record id="hospital_medico_view_tree" model="ir.ui.view">
        <field name="name">hospital.medico.tree</field>
        <field name="model">hospital.medico</field>
        <field name="arch" type="xml">
            <tree>
                <field name="nombre"/>
                <field name="apellidos"/>
                <field name="num_colegiado"/>
            </tree>
        </field>
    </record>

    <record id="hospital_medico_view_form" model="ir.ui.view">
        <field name="name">hospital.medico.form</field>
        <field name="model">hospital.medico</field>
        <field name="arch" type="xml">
            <form string="Médico">
                <group>
                    <field name="nombre"/>
                    <field name="apellidos"/>
                    <field name="num_colegiado"/>
                </group>
            </form>
        </field>
    </record>

    <record id="hospital_medico_action" model="ir.actions.act_window">
        <field name="name">Médicos</field>
        <field name="res_model">hospital.medico</field>
        <field name="view_mode">tree,form</field>
    </record>

</odoo>

```

Código de la view de consulta:

```xml
<?xml version="1.0" encoding="utf-8"?>
<odoo>

    <record id="hospital_consulta_view_tree" model="ir.ui.view">
        <field name="name">hospital.consulta.tree</field>
        <field name="model">hospital.consulta</field>
        <field name="arch" type="xml">
            <tree>
                <field name="paciente_id"/>
                <field name="medico_id"/>
            </tree>
        </field>
    </record>

    <record id="hospital_consulta_view_form" model="ir.ui.view">
        <field name="name">hospital.consulta.form</field>
        <field name="model">hospital.consulta</field>
        <field name="arch" type="xml">
            <form string="Consulta">
                <group>
                    <field name="paciente_id"/>
                    <field name="medico_id"/>
                    <field name="diagnostico"/>
                </group>
            </form>
        </field>
    </record>

    <record id="hospital_consulta_action" model="ir.actions.act_window">
        <field name="name">Consultas</field>
        <field name="res_model">hospital.consulta</field>
        <field name="view_mode">tree,form</field>
    </record>

</odoo>

```

¿Qué tiene de especial?

Guardamos los id de cada modelo involucrado

- paciente_id
- medico_id

Para poder indexar estas vistas, he creado el siguiente xml básico a modo de menu:

```xml
<?xml version="1.0" encoding="utf-8"?>
<odoo>

    <menuitem id="hospital_menu_root" name="Hospital"/>

    <menuitem id="hospital_menu_pacientes"
              name="Pacientes"
              parent="hospital_menu_root"
              action="hospital_paciente_action"/>

    <menuitem id="hospital_menu_medicos"
              name="Médicos"
              parent="hospital_menu_root"
              action="hospital_medico_action"/>

    <menuitem id="hospital_menu_consultas"
              name="Consultas"
              parent="hospital_menu_root"
              action="hospital_consulta_action"/>

</odoo>

```

Configuración de Archivos

Por muy bien que esten definidos nuestros modelos y vistas, no sirve de nada si dejamos de lado el resto de archivos de comunicación con odoo.

Lo primero que debemos hacer, será configurar el __init__.py que se encuentra dentro de la carpeta de models.

Aquí le diremos todos los archivos que odoo debe leer dentro de esta carpeta de la siguiente manera (ya hecho antes):

```
from . import paciente

from . import medico

from . import consulta
```

De la misma manera, en el __init__.py localizado en la raíz del proyecto, debemos registrar que carpeta deberá leer, por lo que colocamos dentro que lea la carpeta de models:

```
from . import models
```

Lo siguiente que he hecho, ha sido configurar el archivo de __manifest__.py, he colocado lo siguiente:

```
{
    "name": "Hospital Gestión",
    "summary": "Gestión de hospital",
    "version": "1.0",
    "category": "Tools",
    "depends": ["base"],
    "data": [
        "security/ir.model.access.csv",
        "views/menus.xml",
        "views/paciente_views.xml",
        "views/medico_views.xml",
        "views/consulta_views.xml",
    ],
    "application": True,
    "installable": True,
}

```

He hecho un poco copia pega del anterior, en el he definido las vistas que hemos creado. El archivo de security lo he cargado a raíz del anterior pero poniendole los campos de este módulo:

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

```
access_hospital_paciente,hospital.paciente,model_hospital_paciente,base.group_user,1,1,1,1

access_hospital_medico,hospital.medico,model_hospital_medico,base.group_user,1,1,1,1

access_hospital_consulta,hospital.consulta,model_hospital_consulta,base.group_user,1,1,1,1
```

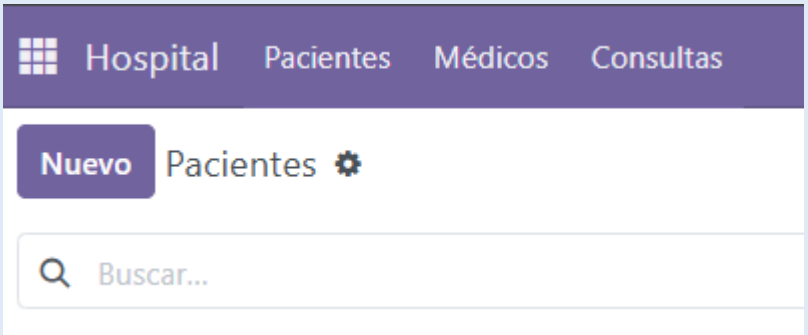
Dándole permisos a los modelos que acabamos de crear.

El siguiente paso que debemos hacer es reiniciar el módulo y forzar el actualizado desde la interfaz de odoo.

Lo realizamos y ya podremos ver lo siguiente:

Ahora sí, haremos click en activar.

El módulo tarda un poco en activarse pero una vez lo podramos usar, nos saldrá de la siguiente manera:



Podremos dar de alta pacientes, médicos y Consultas a través de los forms que hemos creado previamente.

Para demostrar el correcto funcionamiento del módulo vamos a hacer un flujo completo creando dos pacientes que son tendidos por un médico en dos consultas.

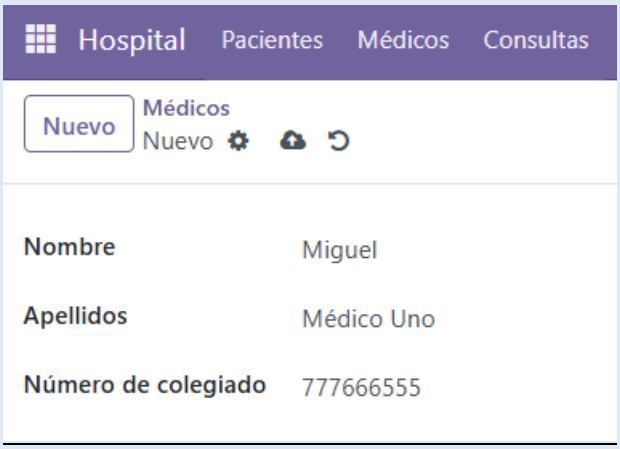
Formulario de alta de pacientes:

Paciente 1:

Paciente 2:

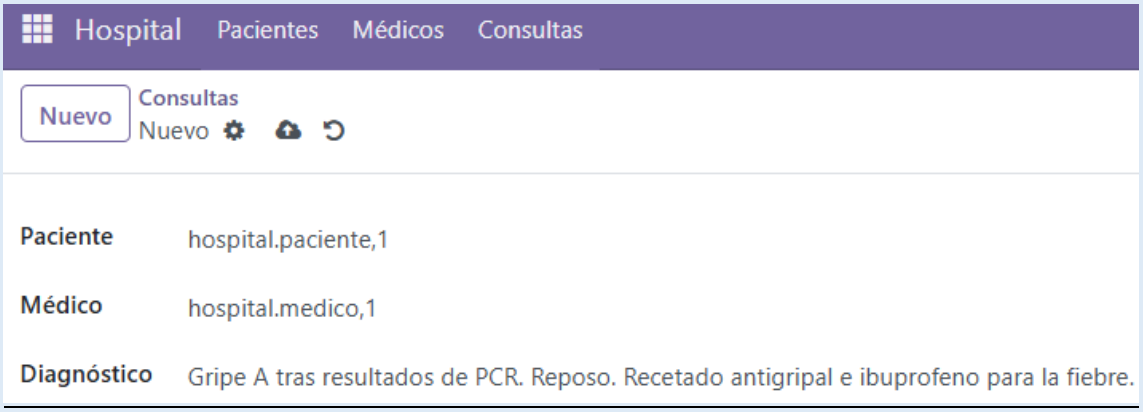
Formulario de alta de médicos:

Médico 1:



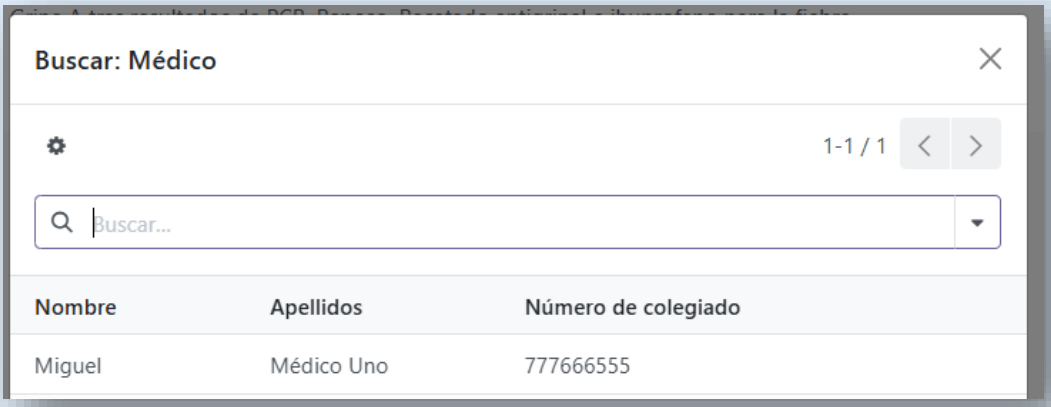
Formulario de alta de consulta:

Consulta a Paciente 1:

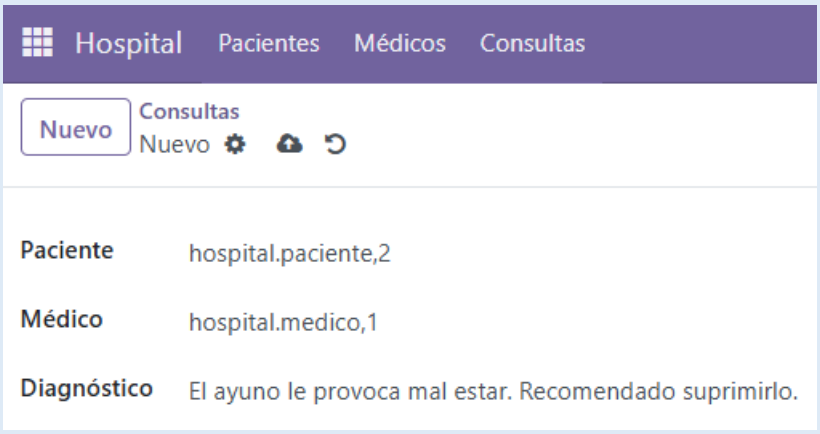


Producido tras seleccionar los identificadores del paciente y del médico.

Es intuitivo y fácil de seleccionar. Esto es un ejemplo de como seleccionas el médico, para los pacientes es exactamente igual:



Esto es un ejemplo del otro paciente siendo atendido por el mismo médico:



Manteniendo las relaciones establecidas por el enunciado del ejercicio. Pasamos a la última actividad.

Actividad 04

Para esta actividad, haré un módulo desde 0 para poder gestionar ciclos formativos. Para ello, creare 4 modelos y las vistas necesarias para su correcta gestión.

Como siempre, lo primero que voy a realizar será la estructuración de la carpeta. No voy a explicar nada de esto ya que es exactamente la misma extructuracion que expliqué en todos los módulos anteriores. Donde mejor lo explico es en la actividad anterior pero en todas las actividades toco algo.

En resumen:

- Dentro de addons creo una carpeta donde alojaré todos los directorios y archivos de mi módulo. Le he llamado a la carpeta “instituto_ciclos/”

Dentro de este directorio, he generado:

- __init__.py
- __manifest__.py
- Carpeta de modelos. Ya he aprovechado y he creado los archivos que creo que voy a usar. Si luego necesito algo más , lo explicaré:
  - __init__.py
  - Ciclo.py
  - Modulo.py
  - Alumno.py
  - Profesor.py
- Carpeta de vistas. Views/
  - Ciclo_views.xml
  - Modulo_views.xml
  - Alumno_views.xml
  - Profesor_views.xml
  - Menus.xml – como hice antes, para poder abrir cada cosa.
- Carpeta de seguridad. Security/
  - Ir.model.access.csv

Esta es la estructura que creo que voy a usar. Como es evidente si has llegado hasta esta parte del documento. Estoy documentando al mismo tiempo que

voy aplicando el desarrollo, me parece algo mas natural, de esta manera es imposible que se me olvide de poner nada. Pero, si luego necesito algo más, no corregiré lo anterior, diré lo que he añadido. Aclarado este punto:

Modelo Ciclo Formativo

Siguiendo el enunciado del ejercicio, el primer módulo que vamos a crear es el del ciclo. Leyendo el resto de modelos, he decidido crearlo de esta manera:

```python
from odoo import models, fields

class InstitutoCiclo(models.Model):
    _name = "instituto.ciclo"
    _description = "Ciclo Formativo"

    name = fields.Char(string="Nombre del ciclo", required=True)

    modulo_ids = fields.One2many(
        "instituto.modulo",
        "ciclo_id",
        string="Módulos"
    )

```

Defino:

- Nombre
- Modulos del ciclo.
    - Un ciclo contiene varios módulos, de ahí la relación de uno a muchos haciendo llamada al string.

En la práctica esto se verá mejor.

Modelo Módulo

He empezado a hacer este, pero creo que es mejor hacer antes alumno y profesor. Paso a desarrollarlos y luego vuelvo a este modelo.

Tras definir los modelos de alumno,ciclo y profesor, he configurado el modelo de módulo de la siguiente manera:

```python
from odoo import models, fields

class InstitutoModulo(models.Model):
    _name = "instituto.modulo"
    _description = "Módulo"

    name = fields.Char(string="Nombre del módulo", required=True)

    ciclo_id = fields.Many2one(
        "instituto.ciclo",
        string="Ciclo formativo",
        required=True
                )

    profesor_id = fields.Many2one(
        "instituto.profesor",
        string="Profesor"
    )

    alumno_ids = fields.Many2many(
        "instituto.alumno",
        "instituto_modulo_alumno_rel",
        "modulo_id",
        "alumno_id",
        string="Alumnos"
    )

```

Tras darle vueltas, he definido lo siguiente:

- Relacion con el ciclo

        - Relación que es de muchos a uno.
            - Cada modulo pertenece a un ciclo
    - Relación con el profesor
        - Muchos a uno
        - Cada módulo puede tener un profesor
    - Relación con alumnos muchos a muchos
        - Un módulo puede tener muchos alumnos

Modelo Alumno

El alumno lo he creado de la siguiente manera:

```python
from odoo import models, fields

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

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

    modulo_ids = fields.Many2many(
        "instituto.modulo",
        "instituto_modulo_alumno_rel",
        "alumno_id",
        "modulo_id",
        string="Módulos"
    )

```

He definido lo siguiente:

- Nombre y apellidos en la misma variable
- Modulo ids
    - Un alumno puede estar en muchos módulos, de ahí que sea N:M

Modelo Profesor

Para el profesor he hecho lo siguiente:

```python
from odoo import models, fields

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

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

    modulo_ids = fields.One2many(
        "instituto.modulo",
        "profesor_id",
        string="Módulos que imparte"
    )

```

He definido:

- Nombre y apellidos en la misma variable
- Modulos que imparte
    - Un profesor puede impartir varios módulos
    - Un módulo es impartido por un profesor. Improtanmte para luego!

Configuración de archivos

¿Vistas?

Para cada todas las vistas he hecho prácticamente lo mismo. Muy repetitivo , mantienen la estructura.

Una vista lista (tree) con un formulario para dar de alta el modelo, asociando cada columna del modelo. Dejo una de ejemplo. Todos los archivos están con exactamente el mismo formato. He creado una vista por cada modelo. CRUD.

A continuación, dejo la vista mas complicada (solo porque llamo a ids pero como en el ejercicio anterior):

```xml
<?xml version="1.0" encoding="utf-8"?>
<odoo>
  <record id="instituto_modulo_tree" model="ir.ui.view">
    <field name="name">instituto.modulo.tree</field>
    <field name="model">instituto.modulo</field>
    <field name="arch" type="xml">
      <tree>
        <field name="name"/>
        <field name="ciclo_id"/>
        <field name="profesor_id"/>
      </tree>
    </field>
  </record>

  <record id="instituto_modulo_form" model="ir.ui.view">
    <field name="name">instituto.modulo.form</field>
    <field name="model">instituto.modulo</field>
    <field name="arch" type="xml">
      <form string="Módulo">
        <group>
          <field name="name"/>
          <field name="ciclo_id"/>
          <field name="profesor_id"/>
        </group>
        <group string="Alumnos matriculados">
          <field name="alumno_ids" widget="many2many_tags"/>
        </group>
      </form>
    </field>
  </record>

  <record id="instituto_modulo_action" model=
"ir.actions.act_window">
    <field name="name">Módulos</field>
    <field name="res_model">instituto.modulo</field>
    <field name="view_mode">tree,form</field>
  </record>
</odoo>

```

Básico, formulario y vista tree.

He configurado el manifest situado en la raíz del repositorio exactamente igual que el resto de módulos:

```
{
    "name": "Ciclos Instituto",
    "summary": "Ciclos, módulos, alumnos y profesores",
    "version": "1.0",
    "category": "Tools",
    "depends": ["base"],
 "data": [
    "security/ir.model.access.csv",
    "views/ciclo_views.xml",
    "views/modulo_views.xml",
    "views/alumno_views.xml",
    "views/profesor_views.xml",
    "views/menus.xml",
],

    "application": True,
    "installable": True,
}

```

Nada a mayores que explicar.

Al igual que anteriores módulos, he declarado en el archivo init de la raíz que debe leer la carpeta de models de la siguiente manera:

```
from . import models
```

Tras esto, el init dentro de la carperta de models, aclara que debemos acceder a cada modelo. Defino lo siguiente en el archivo:

```
from . import ciclo
from . import modulo
from . import alumno
from . import profesor
```

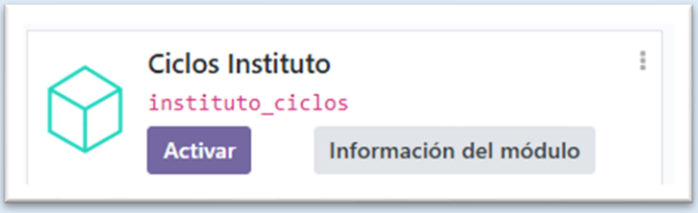
En el archive de seguridad: ir.model.access.csv. Por lo que he leído, luego tendré que modificarlo, luego lo explico al detalle.

He copiado y pegado del ejercicio anterior poniendo los modelos de este ejercicio.

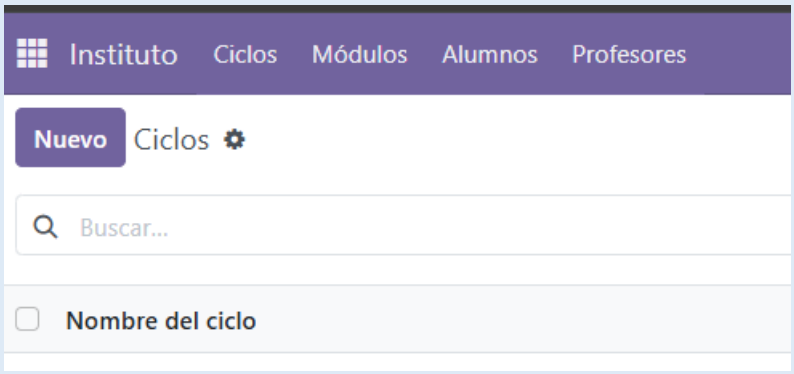
Demostración

Antes de hacer nada de módulos.

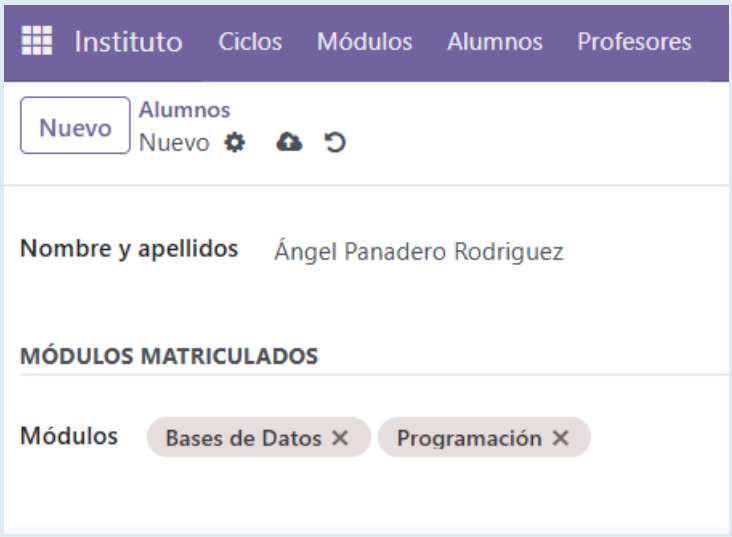
Activamos el modulo:



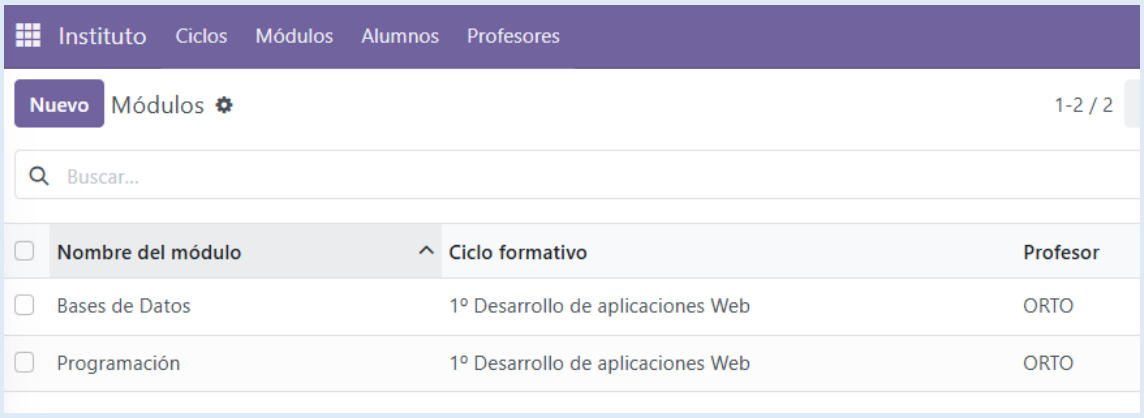
No me ha dado ningún error, la activación se ha completado correctamente:



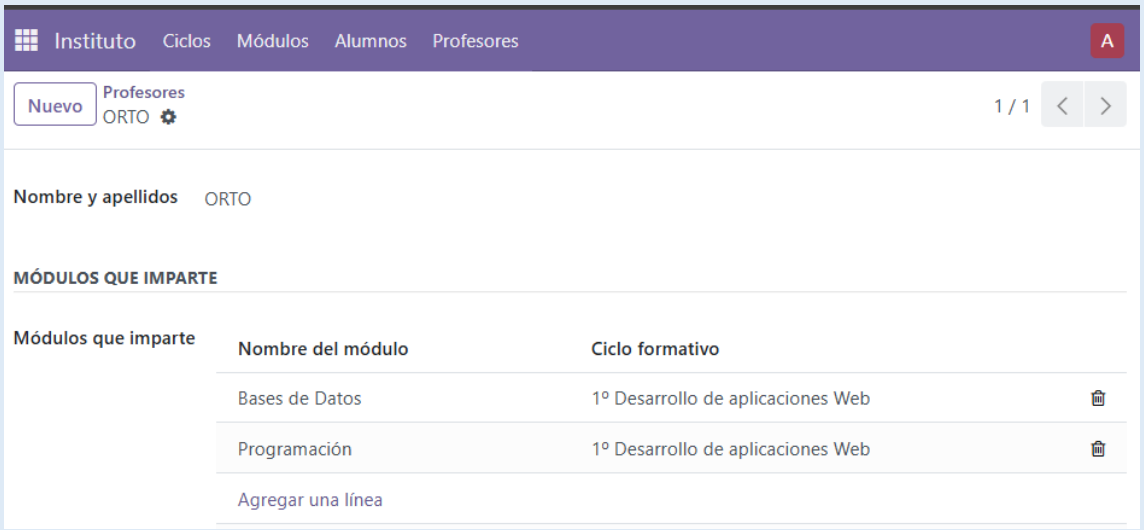
Alumno:



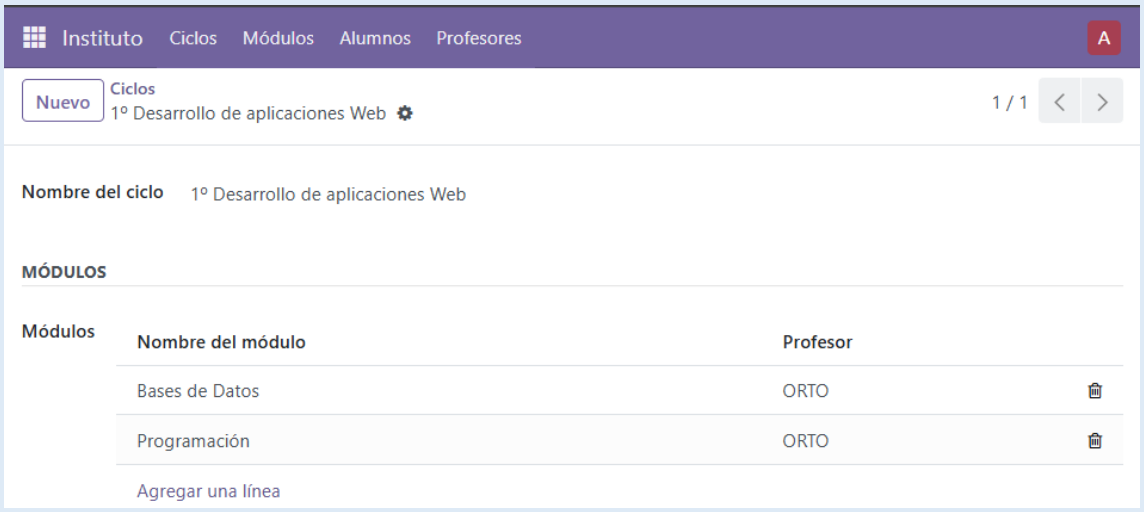
Módulo:



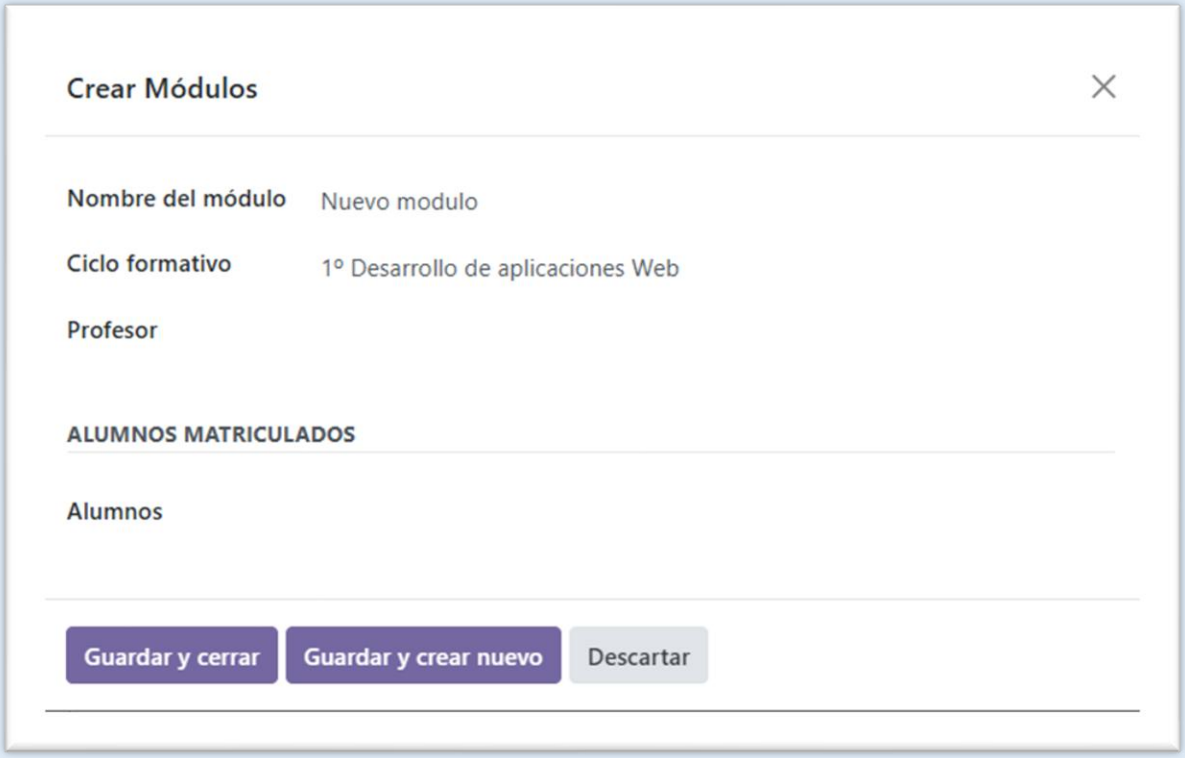
Profesor:



Ciclo:



Todo se relaciona con todo. Las llamadas a los id se generan perfectamente. Incluso he creado algún modelo sobre la marcha:



Seguridad

Vale, me he estado informando y para crear los pasos de seguridad crearé dops grupos de de seguridad:

- Director
    - Acceso total. CRUD
- Profesor
    - Limitado, puede ver profesores pero no tocar
- Resto
    - No podrán ver los profesores.

Para configurarlos, iré de forma individual en ir.model.access.csv para modificarlos.

Dentro de la carpeta security, deberemos crear el siguiente archivo: groups.xml

Dentro, he puesto lo siguiente:

```xml
<?xml version="1.0" encoding="utf-8"?>
<odoo>

  <record id="module_category_instituto" model="ir.module.category">
    <field name="name">Instituto</field>
  </record>

  <record id="group_instituto_director" model="res.groups">
    <field name="name">Director</field>
    <field name="category_id" ref="module_category_instituto"/>
  </record>

  <record id="group_instituto_profesor" model="res.groups">
    <field name="name">Profesor</field>
    <field name="category_id" ref="module_category_instituto"/>
  </record>

</odoo>

```

- Categoria de ajustes, es obligatoria
- Grupo director con id de categoría
- Grupo profesor con id de categoría

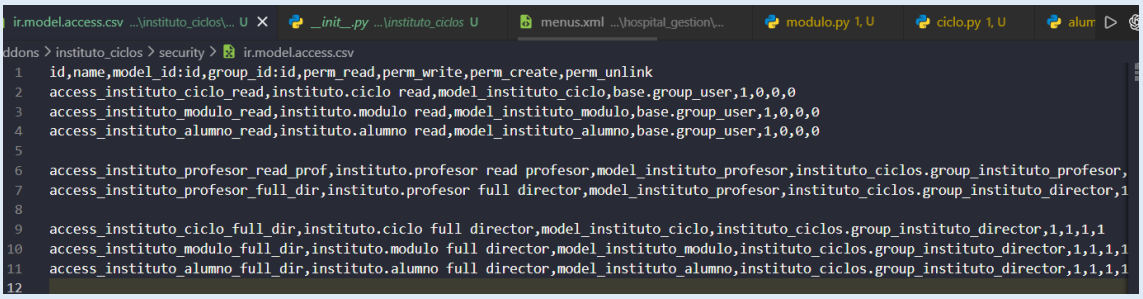
Esto, ya ha creado los roles. Debemos actualizar el data del manifest para que lea los grupos:

```
"security/groups.xml",
```

Lo siguiente que deberemos hacer será modificar el ir.model.access.csv para decir que permisos tienen cada grupo. Lo he puesto asi:

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

access_instituto_ciclo_read,instituto.ciclo read,model_instituto_ciclo,base.group_user,1,0,0,0

access_instituto_modulo_read,instituto.modulo read,model_instituto_modulo,base.group_user,1,0,0,0

access_instituto_alumno_read,instituto.alumno read,model_instituto_alumno,base.group_user,1,0,0,0



access_instituto_profesor_read_prof,instituto.profesor read profesor,model_instituto_profesor,instituto_ciclos.group_instituto_profesor,1,0,0,0

access_instituto_profesor_full_dir,instituto.profesor full director,model_instituto_profesor,instituto_ciclos.group_instituto_director,1,1,1,1



access_instituto_ciclo_full_dir,instituto.ciclo full director,model_instituto_ciclo,instituto_ciclos.group_instituto_director,1,1,1,1

access_instituto_modulo_full_dir,instituto.modulo full director,model_instituto_modulo,instituto_ciclos.group_instituto_director,1,1,1,1

access_instituto_alumno_full_dir,instituto.alumno full director,model_instituto_alumno,instituto_ciclos.group_instituto_director,1,1,1,1
```

En este punto, para que me divierta un poco más, las extensiones de mi VSC han decidido dejar de funcionar, con ella, la extensión que usaba para capturar código por lo que me veo forzado a dejar una cutre foto del código a modo de prueba;



Dentro, he especificado:

- Para ciclo
  - Usuarios normales sin grupo – Solo lectura
- Para profesores
  - Usuarios normales no acceden
  - Profesores leen
  - Director CRUD completo