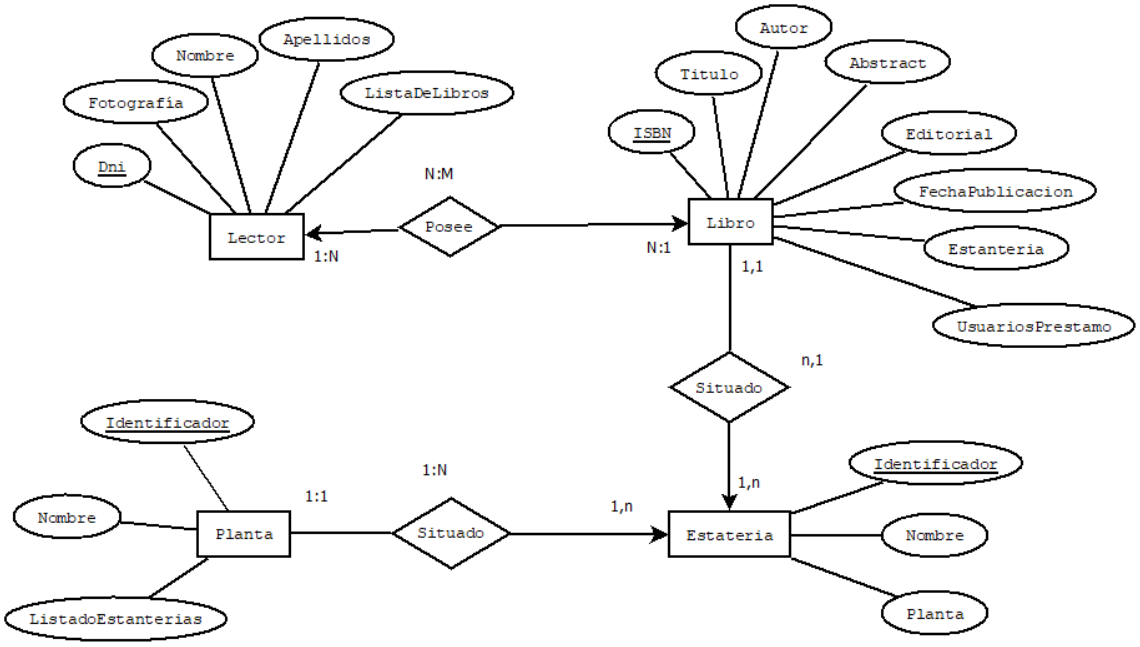
# GESTIÓN BIBLIOTECA CON ODOO


[redacted]

# ÍNDICE

# 1.ESQUEMA ENTIDAD RELACIÓN.

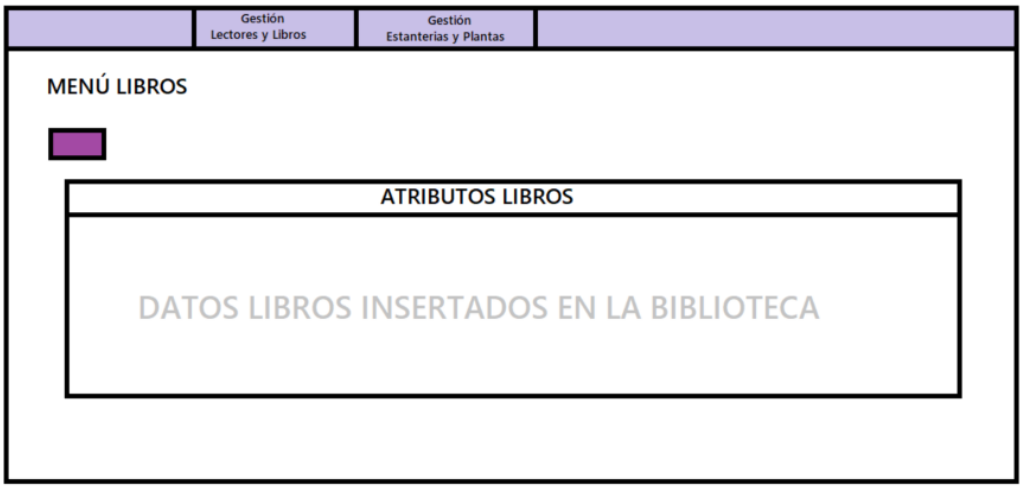



# 2.BOCETO PLATAFORMA.

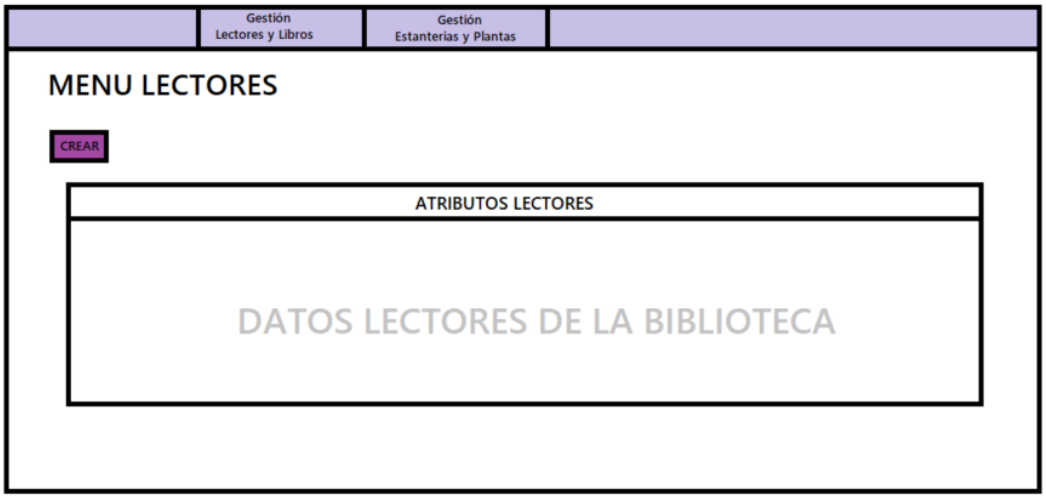
Antes de ponernos a programar debemos saber que es lo que vamos a querer, para ello realizaremos un boceto previo con los menús. El menú principal en Odoo nos mostrara en la barra lateral el ítem Biblioteca. Tal y como se muestra en la siguiente imagen.

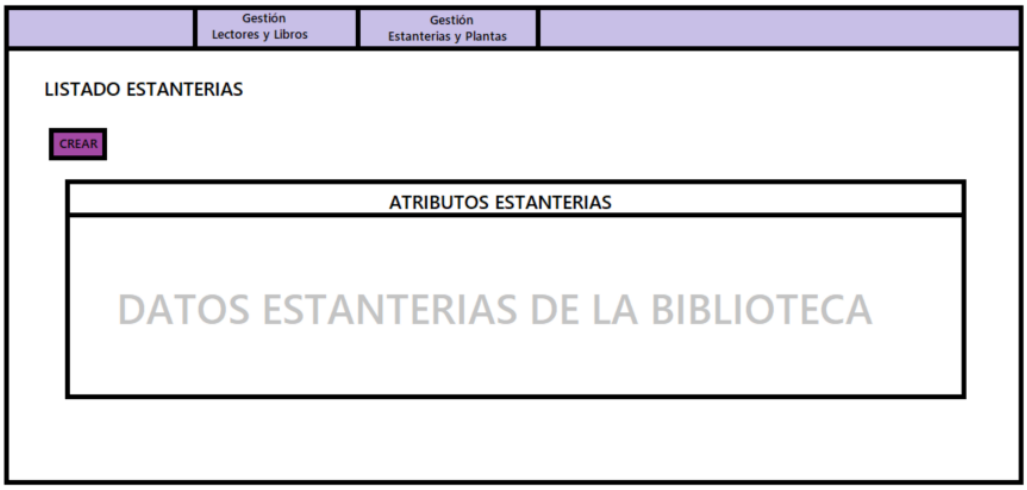
Una vez que hacemos clic en el ítem biblioteca nos aparecerá la siguiente pantalla como principal. Para empezar a tratar los datos e introducirlos en las tablas que posteriormente se crearán en PGADMIN.

Se crearán cuatro pantallas iguales una para cada entidad de nuestro Entidad/Relación que tendrá mínimo un botón de crear. Tal y como se muestran en las siguientes cuatro imágenes.




---







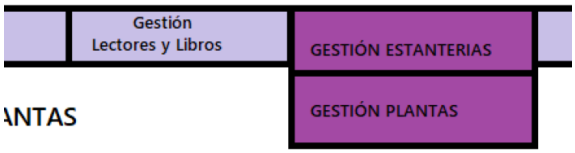
Para acceder a estas cuatro pantallas de las entidades tendremos que crear dos menús que, a su vez, contienen otros dos submenús tal y como se muestra.

Gestión de Lectores y Libros.

- Gestión de Lectores.
- Gestión de Libros

Gestión de estanterías y Plantas.

- Gestión de plantas
- Gestión de estanterías


Cada menú individual nos llevará a una pantalla para introducir los datos que programaremos a continuación en el model.py y en el view.xml

# 3.CÓDIGO DE PROGRAMACIÓN DE LA PLATAFORMA.

Nuestro sistema ERP para la biblioteca nos permitirá a través, del módulo estantería hacer una gestión directa de todos los libros, lectores, plantas y estanterías de formar que estarán relacionados entre sí.

Primero debemos crear el modulo biblioteca con el comando en Ubuntu server, con el comando *"Odoo scaffold biblioteca".*

Una vez creado el módulo nos conectaremos por ssh a nuestro servidor desde visual studio code. Y configuraremos los siguientes archivos de la siguiente manera:

**- Models.py:**

## 3.1.Clase Lector:

```python
# -*- coding: utf-8 -*-

from odoo import models, fields, api


class lectores(models.Model):
    _name = 'biblioteca.lectores'
    _description = 'Gestión de Lectores'
    dni = fields.Char(string='Dni*', required=True, help="Campo obligatorio")
    nombre = fields.Char(string='Nombre')
    apellidos = fields.Char(string='Apellidos')
    fechaNac = fields.Date('Fecha de Nacimiento')
    fotografia=fields.Image(max_width=200, max_height=200)
    libro=fields.Many2many(comodel_name='biblioteca.libro',relation='lector_libro',column1='isbn',column2='dni',string='Libros en prestamo:')
```

### 3.2.Clase Libro:

```
class libro(models.Model):
    _name = 'biblioteca.libro'
    _description = 'Gestión de Libros'
    isbn = fields.Char(string='Isbn *', required=True, help="Campo obligatorio")
    titulo = fields.Char(string="Titulo")
    autor = fields.Char(string="Autor")
    abstract = fields.Binary('Abstract')
    editorial = fields.Char(string='Editorial')
    fechaPub = fields.Date('Fecha de Publicación')
    estanteria=fields.Integer('Numero de estanteria')
    #usuarioPrestamo=fields.Char(string='Usuario en Prestamo')
    estanteria=fields.Many2one('biblioteca.estanteria',cascade=True)

    lectores=fields.Many2many(comodel_name='biblioteca.lectores',relation='lector_libro',column1='dni',column2='isbn',string='Usuarios con el libro
```

### 3.3.Clase Estantería:

```
class estanteria(models.Model):
    _name = 'biblioteca.estanteria'
    _description = 'Gestión de Estanterias'
    identificador = fields.Char(string='identificador', required=True, help="Campo obligatorio")
    nombre = fields.Char(string='nombre')
    plantas=fields.Many2one('biblioteca.planta') #varios libros se asocian a una estanteria
```

### 3.4.Clase Planta:

```
class planta(models.Model):
    _name = 'biblioteca.planta'
    _description = 'Gestión de Plantas'
    identificador = fields.Char(string='identificador', required=True, help="Campo obligatorio")
    listadoEstanterias = fields.Char(string='Nombre Planta')
    nombres = fields.One2many(string='Listados Estanterias', inverse_name='plantas',comodel_name='biblioteca.estanteria')
```

### 3.5.Configuración común de clases:

Todas las clases del *models.py* tienen los mismos elementos comunes.

- _name: Define el nombre del modelo.
- _description: Define la descripción de la clase.
- Atributos:
    - Char: Atributos de texto, serán todos los atributos que necesitan definir algún tipo de nombre o descripción.
    - Integer: Se utilizan para definir numéricamente algún elemento.
    - Binary: En nuestro módulo lo utilizaremos para subir el archivo del abstract ya que es un archivo de texto en pdf.
    - Date: Define la fecha del campo.
    - Image: Utilizado para subir foto, en nuestro caso para la fotografía del lector.

Todos los campos necesitarán una descripción para mostrarla en la interfaz gráfica, tal y como vemos en los puntos anteriores los definiremos después del tipo con el parámetro String=’Nombre campo’.

También podemos utilizar otros elementos auxiliares para el atributo como por ejemplo el required=true, el parámetro help=’ayuda’, que nos definirá un tooltip que ayudará al usuario en algún campo que tenga dudas y el parámetro required=True que nos obligará a que un campo sea obligatorio.

## 3.6.Relaciones de clases:

En nuestro modulo nos encontramos diferentes relaciones tal y como vimos en el esquema E/R. Entre la tabla libros y lectores encontramos una relación Many2Many, entre libro y biblioteca Many2one , entre estantería y plantas Many2One y entre plantas y estanterías one2Many.

**Many2Many:**

Relación en Lectores:

```
libro=fields.Many2many(comodel_name='biblioteca.libro',relation='lector_libro',column1='isbn',column2='dni',string='Libros en prestamo:')
```

Relación en Libro:

```
lectores=fields.Many2many(comodel_name='biblioteca.lectores',relation='lector_libro',column1='dni',column2='isbn',string='Usuarios con el libro:')
```

Esta relación necesita de una tabla auxiliar que se creará en el PGADMIN que más adelante veremos. Para crear esta relación lo primero que definimos es el modelo al que va relacionado en el parámetro comodel_name, después definiremos la nueva relación en relation, las columnas que se relacionan en column1 y column2. (Es importante invertirlas de acuerdo a la relación) y por último mostramos como se verá en la interfaz gráfica.

**Many2One:**

Relación, Libro-Estantería:

```
estanteria=fields.Many2one('biblioteca.estanteria',cascade=True)
```

Esta relación define que una estantería existe muchos libros al igual que en las anteriores lo primero que definimos es el modelo. Y le añadiremos cascade para eliminar todo lo que vaya relacionado con ese campo.

**One2Many:**

Relación Planta – Estantería.

```
nombres = fields.One2many(string='Listados Estanterias', inverse_name='plantas',comodel_name='biblioteca.estanteria')
```

Esta relación define que en una planta existen diferentes estanterías. Con el parámetro string definimos el nombre en la zona visual, inverse_name nos muestra el campo con el que está relacionado de la relación Many2One y finalmente con el comodel_name definimos el modelo al que hemos referenciado.

**-views.xml**

El archivo views.xml será el encargado en relacionar todo lo que tenemos en código phyton en el models.py con la interfaz gráfica de Odoo.

```
iblioteca > views > views.xml
<odoo>
  <data>
    <record model="ir.ui.view" id="biblioteca.lectores_list_view">
      <field name="name">biblioteca.lectores.view.tree</field>
      <field name="model">biblioteca.lectores</field>
      <field name="arch" type="xml">
        <tree>
          <field name="dni"/>
          <field name="nombre"/>
          <field name="apellidos"/>
          <field name="fechaNac"/>
          <field name="fotografia"/>
        </tree>
      </field>
    </record>

<record model="ir.ui.view" id="biblioteca.libro_list_view">
      <field name="name">biblioteca.libro.view.tree</field>
      <field name="model">biblioteca.libro</field>
      <field name="arch" type="xml">
        <tree>
          <field name="isbn"/>
          <field name="titulo"/>
          <field name="autor"/>
          <field name="abstract"/>
          <field name="editorial"/>
          <field name="fechaPub"/>
         <field name="estanteria"/>
        </tree>
      </field>
    </record>
```

```

<record model="ir.ui.view" id="biblioteca.estanteria_list_view">
      <field name="name">biblioteca.estanteria.view.tree</field>
      <field name="model">biblioteca.estanteria</field>
      <field name="arch" type="xml">
        <tree>
          <field name="identificador"/>
          <field name="nombre"/>
          <field name="plantas"/>
        </tree>
      </field>
    </record>

<record model="ir.ui.view" id="biblioteca.planta_list_view">
      <field name="name">biblioteca.planta.view.tree</field>
      <field name="model">biblioteca.planta</field>
      <field name="arch" type="xml">
        <tree>
          <field name="identificador"/>
          <field name="listadoEstanterias"/>
           <field name="nombres"/>
        </tree>
      </field>
    </record>


```

En el XML del view debemos definir nuestra id, primeramente. Para ello pondremos en la id=biblioteca.nombreModelo_list_view.

Definimos el nombre en name=Nombre modelo, model=Modelo y el arch=tipo XML

En el árbol de jerarquía definimos los campos tree y field donde vamos a definir visualmente todos los campos.  Todos los modelos serán iguales tal y como se han mostrado en las imágenes anteriores.

## 3.7 Creación de menús y submenús.

Para crear los menús y submenús de nuestra interfaz gráfica dentro del view.xml creamos lo siguiente:

**Actions:**

```
<!-- actions opening views on models -->
 <record model="ir.actions.act_window" id="biblioteca.lectores_action_window">
  <field name="name">Lectores de la Biblioteca</field>
  <field name="res_model">biblioteca.lectores</field>
  <field name="view_mode">tree,form</field>
 </record>

<record model="ir.actions.act_window" id="biblioteca.libro_action_window">
  <field name="name">Libros de la Biblioteca</field>
  <field name="res_model">biblioteca.libro</field>
  <field name="view_mode">tree,form</field>
 </record>

 <record model="ir.actions.act_window" id="biblioteca.estanteria_action_window">
  <field name="name">Estanterias </field>
  <field name="res_model">biblioteca.estanteria</field>
  <field name="view_mode">tree,form</field>
 </record>

 <record model="ir.actions.act_window" id="biblioteca.planta_action_window">
  <field name="name">Plantas </field>
  <field name="res_model">biblioteca.planta</field>
  <field name="view_mode">tree,form</field>
 </record>
```

Los actions definen la relación con el menú al abrirlo le tenemos que dar un nombre y configurarlo tal y como se muestra en la imagen asignándole el id correspondiente.

**Menú principal:**

```
    <!-- Top menu item -->
<menuitem name="Biblioteca" id="biblioteca.menu_root"/>
    <!-- menu categories -->
```

Definimos el nombre del menú y lo referenciamos al padre en este caso es el root.

**Submenú:**

Para nuestra plataforma vamos a crear dos menús referenciados al padre con el parámetro parent= al root. Y en el id le asignamos dos id diferentes que en el siguiente paso utilizaremos.

Menú Lectores y Libros = menu_1.

Menú Plantas y estanterías= menú_2.

```
    <!-- menu categories -->

<menuitem name="Menu Lectores y Libros" id="biblioteca.menu_1" parent="biblioteca.menu_root"/>
<menuitem name="Menu Plantas y Estanterias" id="biblioteca.menu_2" parent="biblioteca.menu_root"/>
```

**Menús específicos:**

Dentro de cada submenú hemos creados cuatro menús

Menú Lectores y Libros.

Menú Gestión Lector:menu_1

Menú Gestión Libros: menu_1

Menú Plantas y Estanterías.

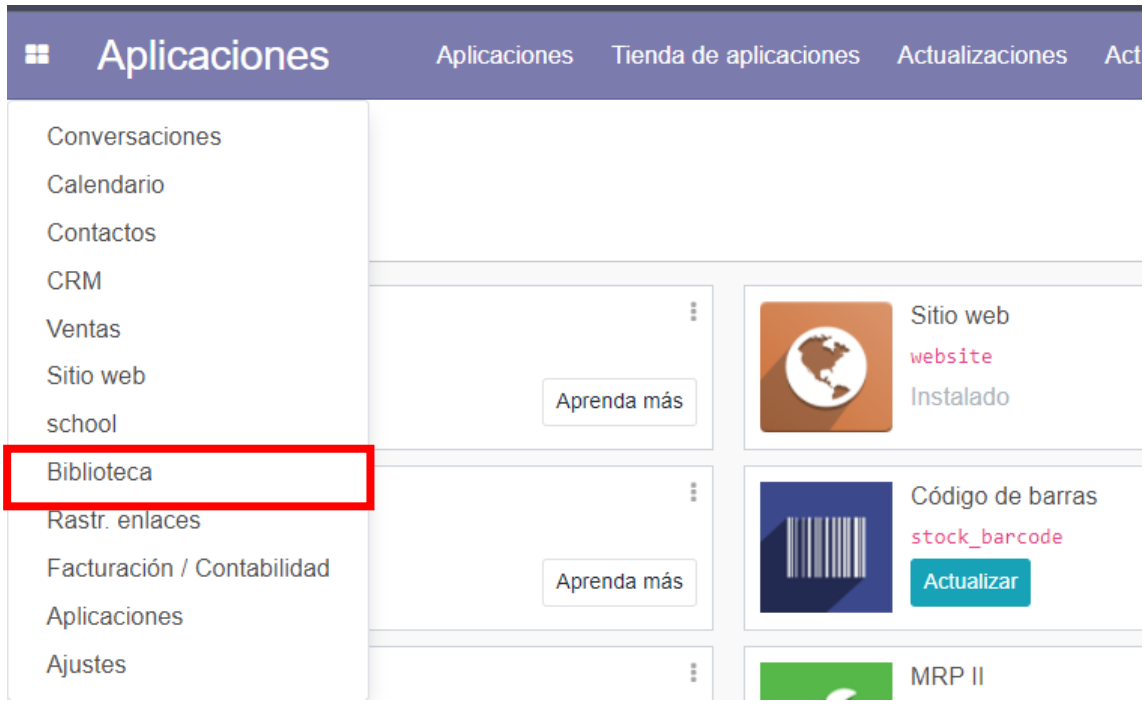
Menú Gestión Estantería: menu_2

Menú Gestión Plantas:menu_2

```
<!-- actions -->
    <menuitem name="Gestión Lectores" id="biblioteca.menu_lectores" parent="biblioteca.menu_1"
    action="biblioteca.lectores_action_window"/>
   <menuitem name="Gestión Libros" id="biblioteca.menu_libro" parent="biblioteca.menu_1"
    action="biblioteca.libro_action_window"/>
    <menuitem name="Gestión Estanterias" id="biblioteca.menu_estanteria" parent="biblioteca.menu_2"
    action="biblioteca.estanteria_action_window"/>
    <menuitem name="Gestión Plantas" id="biblioteca.menu_planta" parent="biblioteca.menu_2"
    action="biblioteca.planta_action_window"/>
```

# 4.FUNCIONALIDAD DE LA PLATAFORMA.

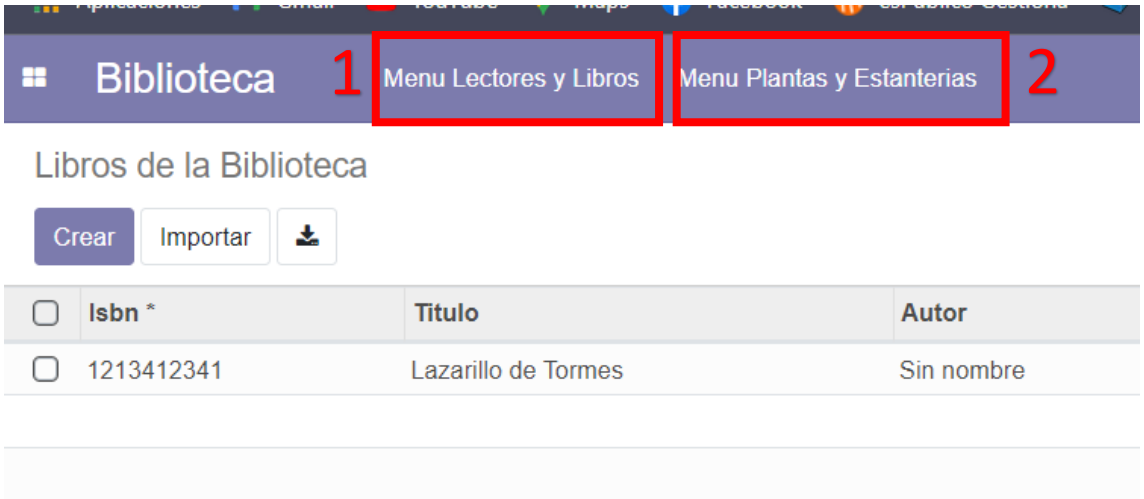
Basándonos en el boceto de la plataforma y una vez que hemos actualizado nuestra aplicación desde el menú aplicaciones, biblioteca. Nos debe aparecer el menú biblioteca. El cual su padre era la pantalla principal root.



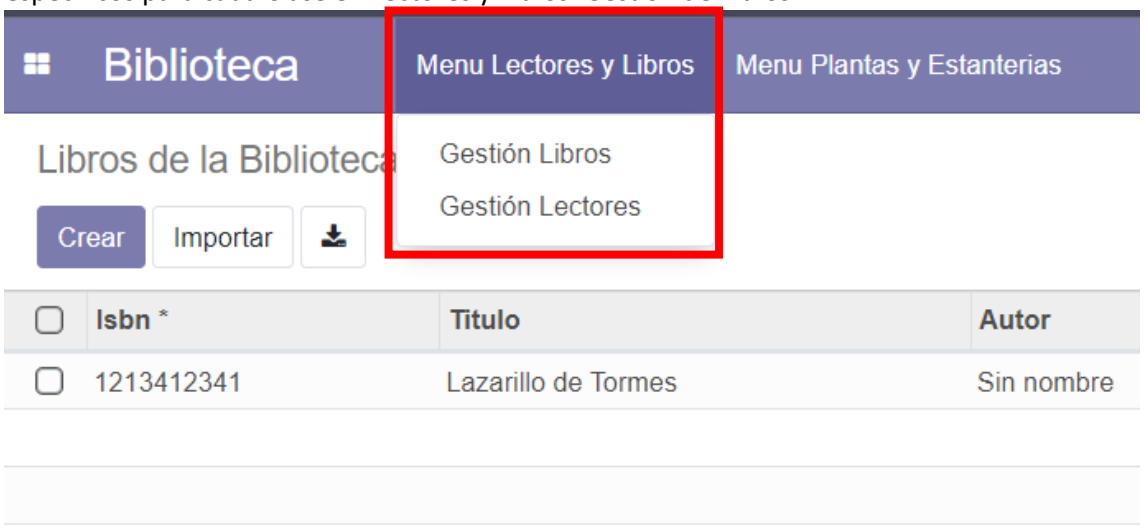

Una vez que hemos entrado en Biblioteca tendremos creado los dos submenús.

En la parte del view que mostramos anteriormente hemos añadido dos submenús que nombramos con el id tal que así:

1. Menu_1
2. Menu_2



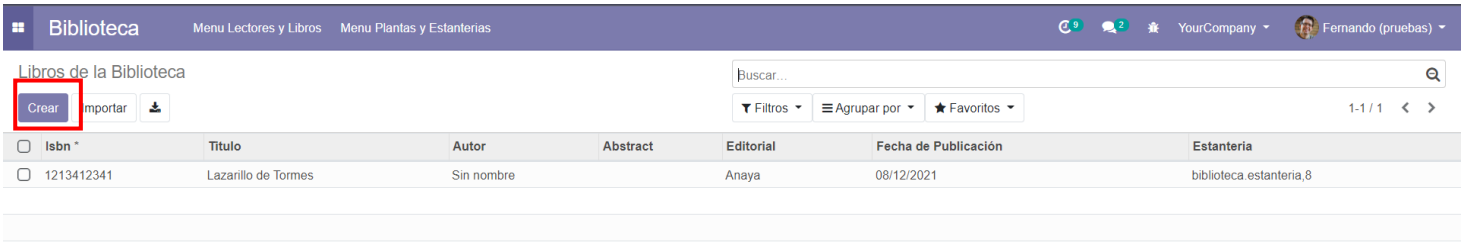
Desplegamos el menú Lectores y libros, y nos aparecerán otros dos submenús específicos para cada Clase en Lectores y Libros. Gestión de Libros




Y si desplegamos el menú Plantas y Estanterías nos mostrará, Gestión de plantas y Gestión de Libros

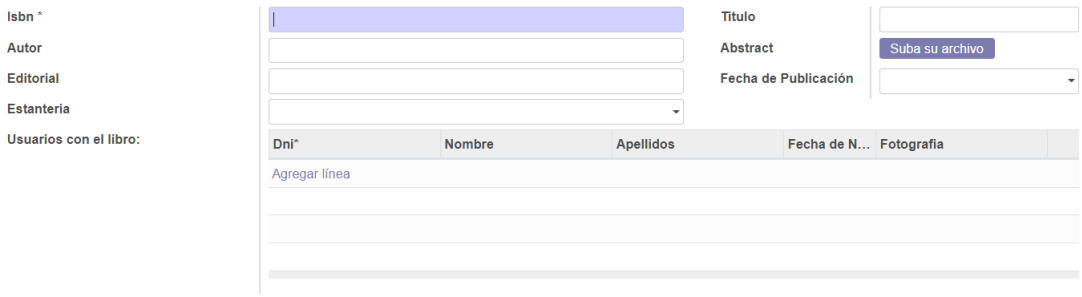
## 4.1 Gestión de Libros

A continuación, se muestra la vista principal con todos los atributos que le asignamos en el views.xml si pinchamos en la opción Crear nos mostrará el menú de la imagen 2.



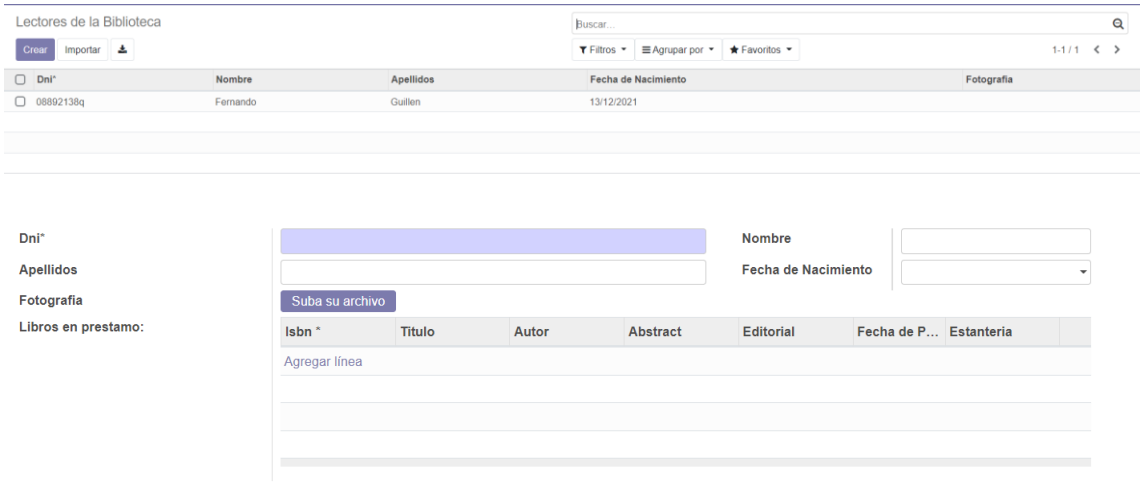

Una vez que pulsamos en el botón crear nos aparecerá el menú que definimos en el action junto con el view. Donde se muestran todos los campos que tiene la tabla Libros.

En la parte inferior nos muestra una pequeña pantalla donde podremos relacionar los usuarios que tienen ese libro en préstamo, ya que en el models.py hicimos la relación de Many2Many.



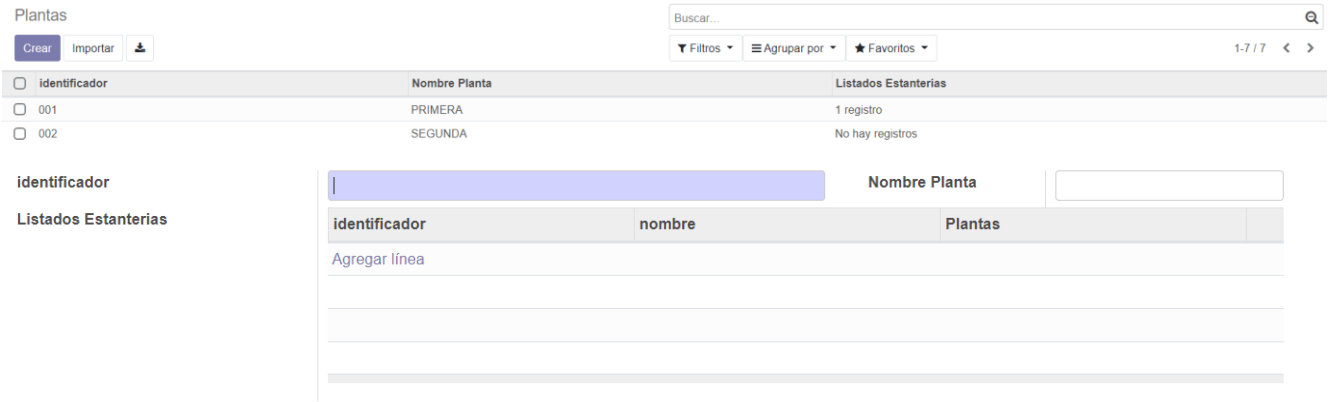
## 4.2 Gestión de Lectores

Este menú es muy parecido al menú de Libros donde al crear un lector pulsando en el botón crear , nos aparecerá una pantalla para introducir todos sus campos y gracias a la relación many2many una pantalla para relacionar todos los libros que este usuario tiene en préstamo.




## 4.3 Gestión de Plantas

El menú principal de la gestión de plantas es similar a los anteriores , nos aparecerá un identificador para hacer un control interno, el nombre de la planta y un Listado de las estanterías que tienen estas plantas. Al pulsar el botón crear podremos agregar estas estanterías ya que hicimos una relación One2Many es decir una planta puede albergar una o muchas estanterías.



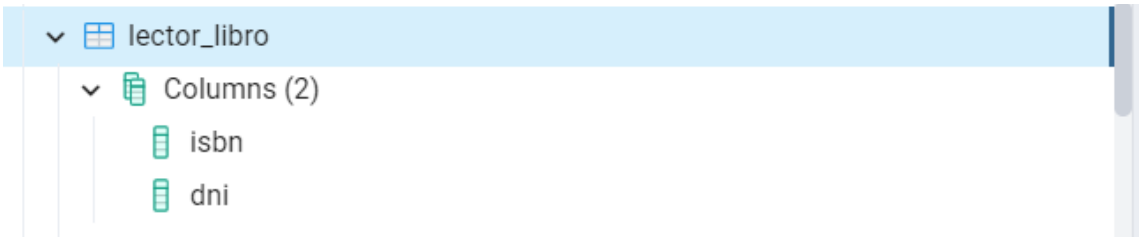
## 4.4 Gestión de Estanterías

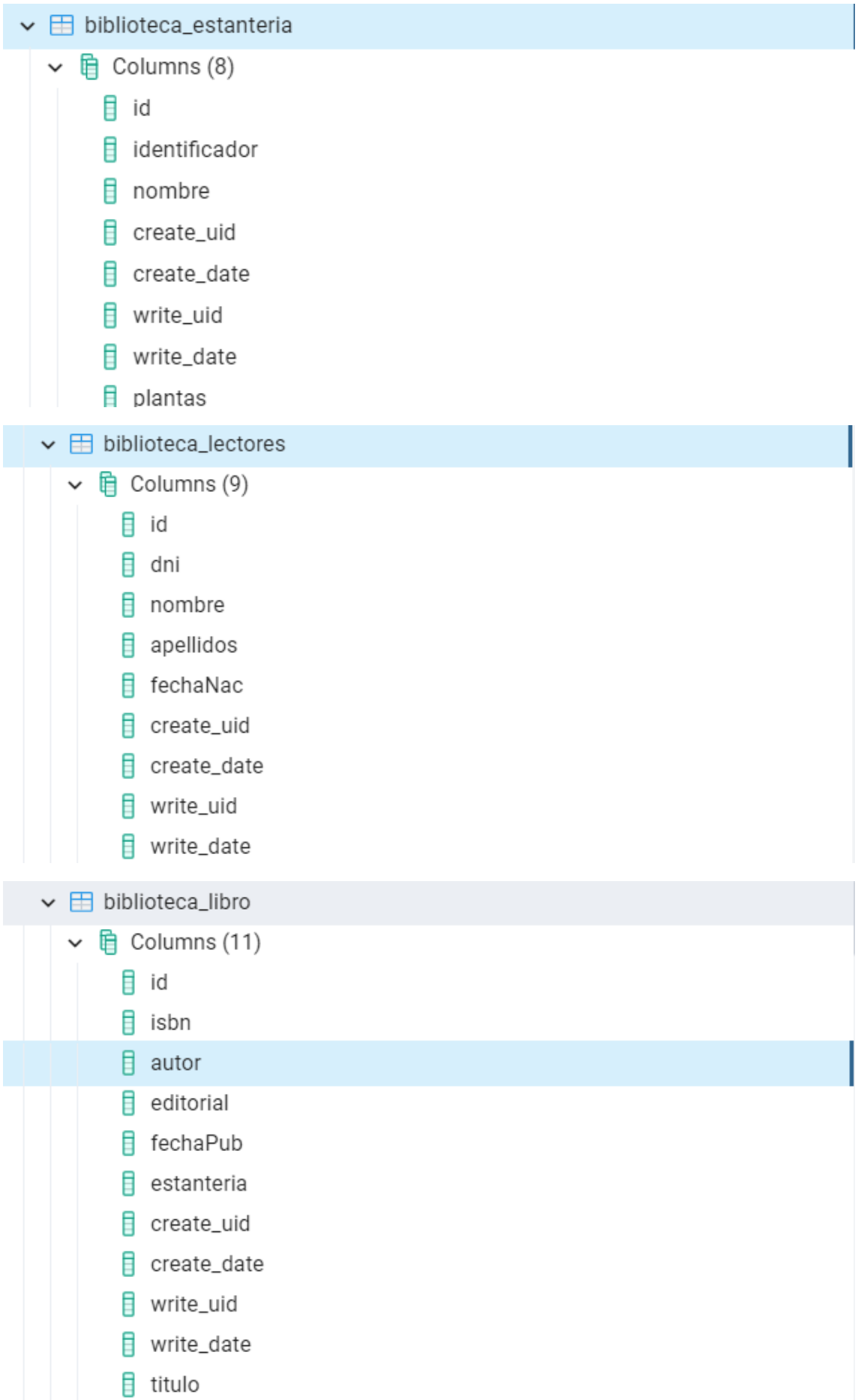
Por último, en el menú de estanterías , nos encontramos la vista estantería , donde nos aparecerá la vista con los datos de las estanterías y en que planta está relacionada. Si pinchamos en crear podremos crear una estantería y añadirla a una planta gracias a la relación Many2One donde muchas estanterías pueden estar en una planta.



# 5.VOLCADO PGADMIN

PgAdmin nos creará de forma automática las tablas que hemos creado con visual studio code en el models.py y en las views. Así como la tabla auxiliar de las relaciones. Para ello nos vamos a nuestra base de datos pruebas y en tables buscamos el nombre que le dimos a cada tabla en los apartados anteriores. Y tal y como se muestra nos ha creado exactamente todas las tablas con sus diferentes columnas, así como columnas auxiliares que utiliza el pgAdmin para su gestión interna.




biblioteca_estanteria
Columns (8)
id
identificador
nombre
create_uid
create_date
write_uid
write_date
plantas
biblioteca_lectores
Columns (9)
id
dni
nombre
apellidos
fechaNac
create_uid
create_date
write_uid
write_date
biblioteca_libro
Columns (11)
id
isbn
autor
editorial
fechaPub
estanteria
create_uid
create_date
write_uid
write_date
titulo


biblioteca_planta
Columns (8)
id
identificador
create_uid
create_date
write_uid
write_date
listadoEstanterias
nombre