

Ramos Personalizados

(Gestor de creación de ramos personalizados y sus envíos)

Índice de contenidos

1. Resumen (1.5 punto).....	1
2. Estructura del proyecto (0.5 punto).....	1
3. Contenido de cada fichero (1 punto).....	2
4. Implementación de dos modelos (1 punto).....	12
5. Creación de más modelos (1 punto).....	13
6. Creación de las vistas, modelos y menús dentro de Odoo (0.5 punto).....	13
7. Modificación de las vistas (1.5 punto).....	15
8. Restricciones en los campos (1.5 punto).....	20
.....	20
9. pgAdmin (0.5 punto).....	22

Índice de figuras

Figura 1: Estructura del módulo.....	1
Figura 2: models/__init__.py.....	2
Figura 3: models/Flor.py.....	3
Figura 4: models/Pedido.py.....	3
Figura 5: models/Ramo.py.....	4
Figura 6: models/PedidoRamo.py.....	5
Figura 7: models/RamoFlor.py.....	5
Figura 8: security/ir.model.access.csv.....	6
Figura 9: views/ViewFlor.xml.....	7
Figura 10: views/ViewPedido.xml.....	9
Figura 11: views/ViewRamo.xml.....	11
Figura 12: __init__.py.....	12
Figura 13: __manifest__.py.....	12
Figura 14: Ramos dentro de odoo.....	13
Figura 15: Pedidos dentro de odoo.....	14
Figura 16: Flores dentro de odoo.....	14
Figura 17: Código de ViewFlor.....	15
Figura 18: código de ViewPedido.....	17
Figura 19: Código de ViewRamo.....	19
Figura 20: Muestra de que la restricción funciona.....	20
Figura 21: Código de flor para ver la restricción.....	20
Figura 22: Imagen de pgadmin de las tablas creadas.....	22

1. Resumen (1.5 punto)

El módulo Ramos Personalizados se encarga de gestionar la creación y venta de ramos personalizados.

Como usuario, puedes agregar nuevas flores a la base de datos, especificando su nombre, color y precio (el cual no puede ser negativo). Estas flores pueden añadirse a distintos ramos, indicando la cantidad deseada. El precio del ramo se calcula automáticamente en función de las flores seleccionadas. Además, cada ramo puede recibir un nombre para facilitar su identificación.

Por último, el módulo incluye un apartado de pedidos, donde se registran los ramos a vender junto con sus cantidades. Cada pedido es identificado por un nombre y se asocia a un empleado responsable de su gestión.

2. Estructura del proyecto (0.5 punto)

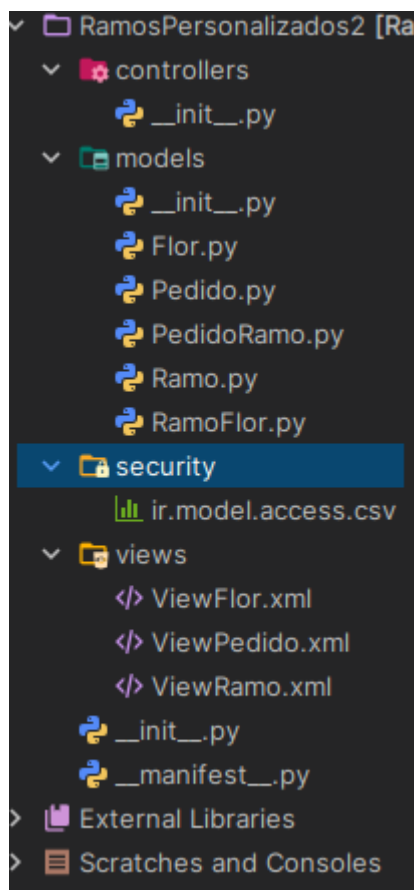


Figura 1: Estructura del módulo

El modulo contiene 4 carpetas aparte del, init y el manifest:
controllers → aquí no tengo nada

Por:David Sánchez Peso

models → aquí tengo los modelos del modulo (Flor,Pedido,PedidoRamo,Ramo,RamoFlor)

security → aquí tengo el archivo de seguridad (ir.model.access.csv)

views → aquí tengo todas las vistas del modulo en xml (ViewFlor,ViewPedido y ViewRamo)

3. Contenido de cada fichero (1 punto)

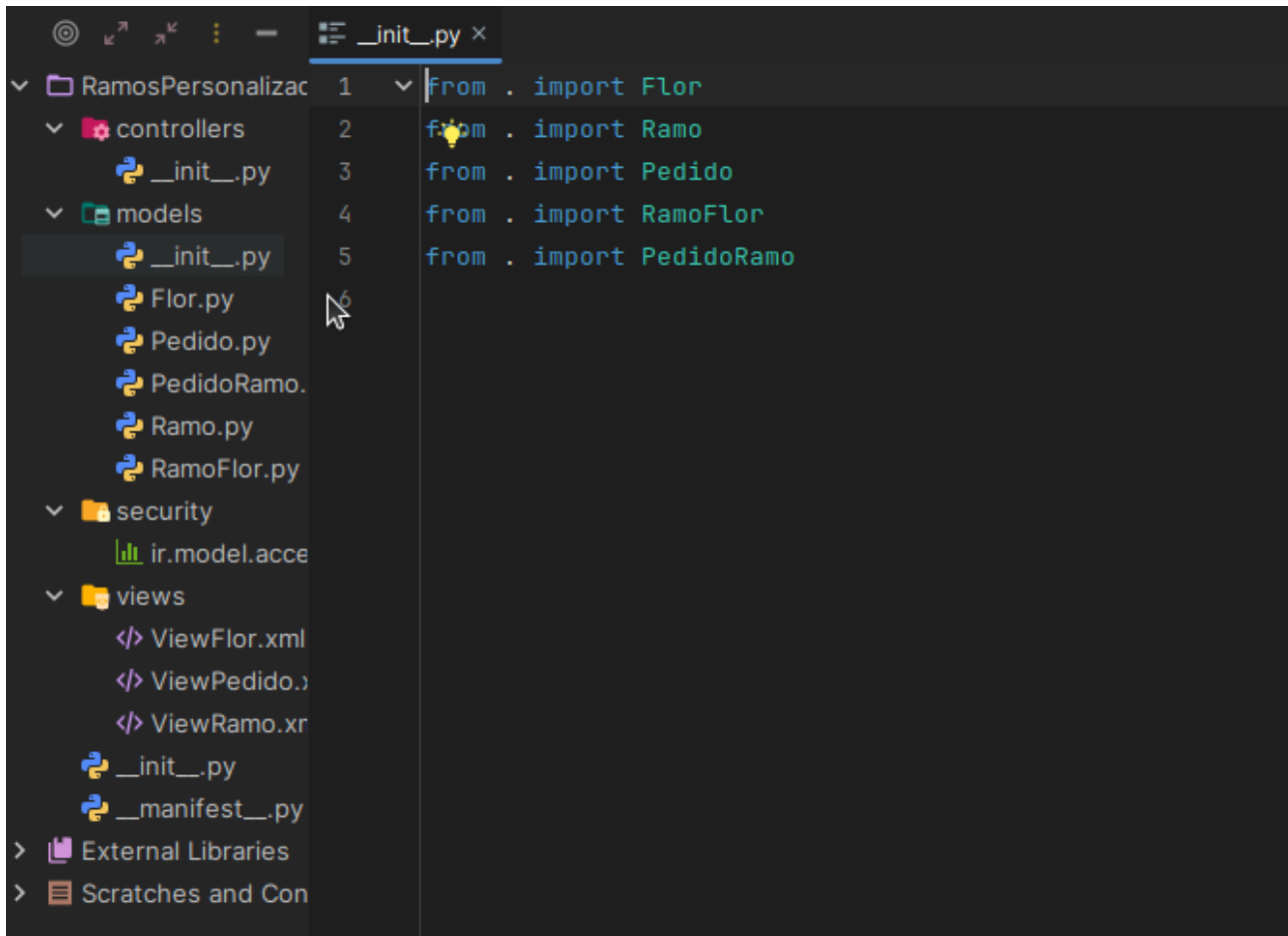


Figura 2: models/__init__.py

El __init__.py contiene los imports para que odoo encuentre los modelos del modulo

```
from odoo import api, fields, models

# Modelo Flor
class Flor(models.Model):
    _name = "ramospersonalizados.flor"
    _description = "Flores"
    _rec_name = "name"

    name = fields.Char(string="Nombre de la flor", required=True)
    color = fields.Char(string="Color")
    precio = fields.Float(string="Precio por unidad", required=True) # Precio de cada flor

    _sql_constraints = [
        ('check_precio_no_negativo', 'CHECK(precio >= 0)', 'El precio no puede ser negativo.')
    ]
```

Figura 3: models/Flor.py

este es el primer modelo que contiene:

name = nombre de la flor

color = el color de la flor

precio = el precio de la unidad de flor

_sql_constraint = una restricción para que el precio no sea negativo

_name = el nombre de la tabla que se creará

_description = descripción del módulo

_rec_name = qué campo se usará al mostrarse al ser usado en otro modelo en la parte gráfica

```
from odoo import api, fields, models

class Pedido(models.Model):
    _name = "ramospersonalizados.pedido"
    _description = "Pedidos de Ramos Personalizados"

    name = fields.Char(string="Nombre del Pedido", required=True)
    empleado_id = fields.Many2one('hr.employee', string="Empleado Responsable") # Relación con el empleado
    ramo_ids = fields.One2many('ramospersonalizados.pedidoramo', 'pedido_id', string="Ramos") # Relación con los ramos
    total = fields.Float(string="Total", compute="_compute_total", store=True) # Total del pedido calculado

    @api.depends('ramo_ids.precio_total') # Dependiendo del precio total de los ramos y las cantidades
    def _compute_total(self):
        for pedido in self:
            pedido.total = sum(ramo.precio_total for ramo in pedido.ramo_ids) # Sumar todos los precios de los ramos
```

Figura 4: models/Pedido.py

Por:David Sánchez Peso

guarda los ramos del pedido, precio total y empleado que se encarga del pedido

_name = el nombre de la tabla que se creara

_description = descripción del módulo

name = nombre del pedido

empleado_id = empleado encargado del pedido

ramos_ids = los id de los ramos del pedido (va a la tabla intermedia)

total = el precio total del pedido según los ramos seleccionados

_compute_total = calcula el total del pedido según cantidad y ramo

```
1 from odoo import api, fields, models
2
3 class Ramo(models.Model):
4     _name = "ramospersonalizados.ramo"
5     _description = "Ramos personalizados"
6     _rec_name = "name"
7
8     name = fields.Char(string="Nombre del ramo", required=True)
9     precio = fields.Float(string="Precio", compute="_compute_precio", store=True) # Precio total calculado
10    flores_ids = fields.One2many('ramospersonalizados.ramoflor', 'ramo_id', string="Flores") # Relación con flores
11
12    @api.depends('flores_ids.precio_total') # Dependiendo del precio de las flores y sus cantidades
13    def _compute_precio(self):
14        for ramo in self:
15            ramo.precio = sum(flor.precio_total for flor in ramo.flores_ids) # Sumar todos los precios de las flores
16
```

Figura 5: models/Ramo.py

guarda las flores que van en cada ramos calculando el precio total según las flores y su cantidad y un nombre para identificarlo

_name = el nombre de la tabla que se creara

_description = descripción del módulo

_rec_name = que campo se usara al mostrarse al ser usado en otro modelo en la parte grafica

name = el nombre del ramo

precio = el precio autocalculado total del ramo

flores_ids = las flores que compone el ramo (va a la tabla intermedia)

_compute_precio = calcula el precio total del ramo

```
from odoo import api, fields, models # Importación correcta

class PedidoRamo(models.Model):
    _name = "ramospersonalizados.pedidoramo"
    _description = "Relación entre Pedido y Ramo"

    pedido_id = fields.Many2one('ramospersonalizados.pedido', string="Pedido", required=True) # Relación con pedido
    ramo_id = fields.Many2one('ramospersonalizados.ramo', string="Ramo", required=True) # Relación con ramo
    cantidad = fields.Integer(string="Cantidad", default=1, required=True) # Cantidad de ramos en el pedido
    precio_total = fields.Float(string="Precio Total", compute="_compute_precio_total", store=True) # Precio total de los ramos en el pedido

    @api.depends('ramo_id.precio', 'cantidad') # Dependiendo del precio del ramo y la cantidad 1usage
    def _compute_precio_total(self):
        for record in self:
            record.precio_total = record.ramo_id.precio * record.cantidad # Calcular el precio total de los ramos en el pedido
```

Figura 6: models/PedidoRamo.py

_name = Nombre de la tabla que se creará para almacenar los pedidos.

_description = Descripción del módulo.

_rec_name = Campo que se usará para mostrar el pedido en la interfaz gráfica.

name = Nombre del pedido para su identificación.

empleado_id = Referencia al empleado encargado del pedido.

ramos_ids = Lista de ramos incluidos en el pedido (relación con la tabla intermedia).

total_precio = Precio total del pedido, calculado automáticamente.

_compute_total_precio = Método para calcular el precio total del pedido sumando los precios de los ramos incluidos.

```
from odoo import api, fields, models

class RamoFlor(models.Model):
    _name = "ramospersonalizados.ramoflor"
    _description = "Relación entre Ramo y Flor"

    ramo_id = fields.Many2one('ramospersonalizados.ramo', string="Ramo", required=True) # Relación con ramo
    flor_id = fields.Many2one('ramospersonalizados.flor', string="Flor", required=True) # Relación con flor
    cantidad = fields.Integer(string="Cantidad", default=1, required=True) # Cantidad de flores
    precio_total = fields.Float(string="Precio Total", compute="_compute_precio_total", store=True) # Precio total de las flores en el ramo

    @api.depends('flor_id.precio', 'cantidad') # Dependiendo del precio de la flor y la cantidad 1usage
    def _compute_precio_total(self):
        for record in self:
            record.precio_total = record.flor_id.precio * record.cantidad # Calcular el precio total
```

Figura 7: models/RamoFlor.py

_name = Nombre de la tabla intermedia que relaciona ramos y flores.

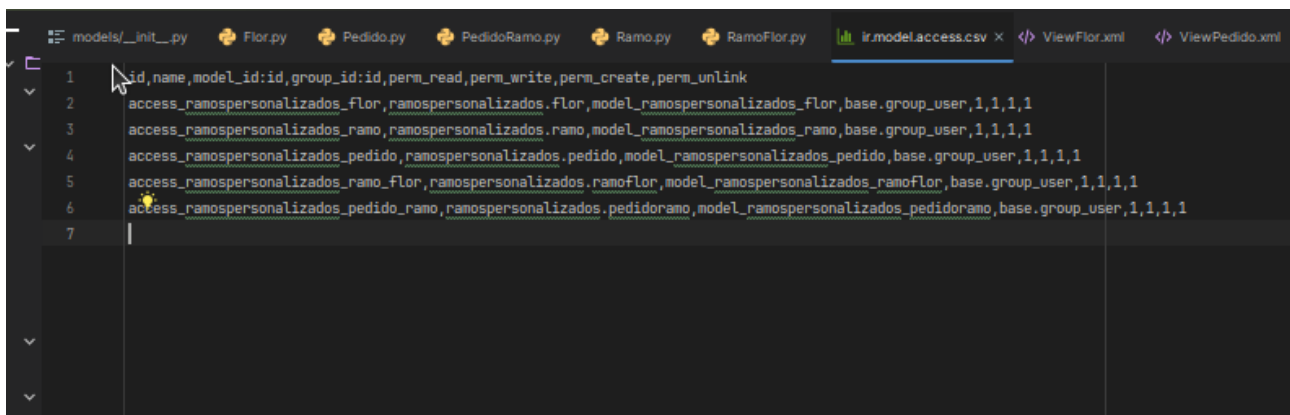
_description = Descripción del módulo.

ramo_id = Referencia al ramo al que pertenece la flor.

flor_id = Referencia a la flor que forma parte del ramo.

Cantidad = Número de unidades de esa flor dentro del ramo

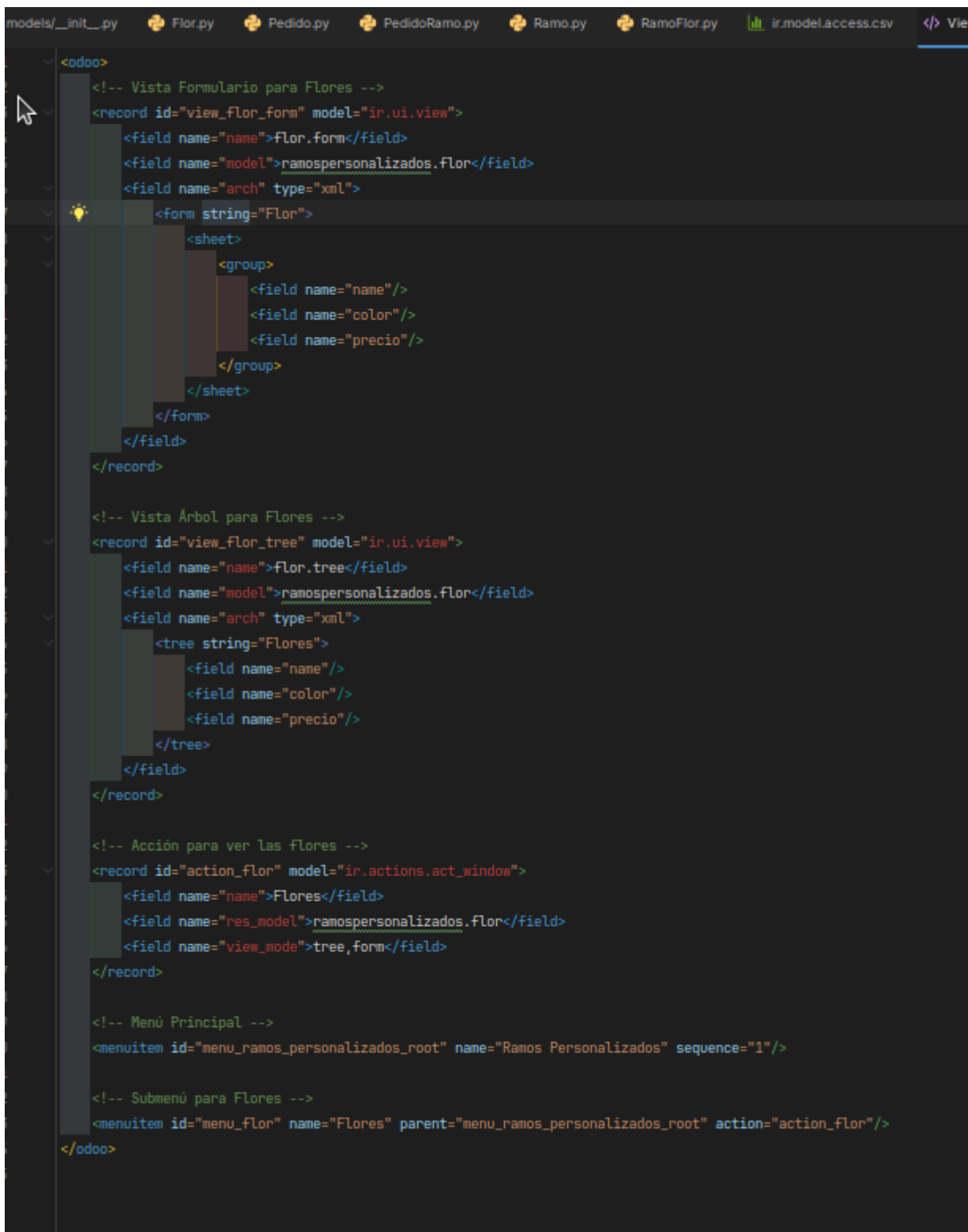
Por:David Sánchez Peso



```
1 id,name,model_id:id,group_id:id,perm_read,perm_write,perm_create,perm_unlink
2 access_ramospersonalizados_flor,ramospersonalizados.flor,model_ramospersonalizados_flor,base.group_user,1,1,1,1
3 access_ramospersonalizados_ramo,ramospersonalizados.ramo,model_ramospersonalizados_ramo,base.group_user,1,1,1,1
4 access_ramospersonalizados_pedido,ramospersonalizados.pedido,model_ramospersonalizados_pedido,base.group_user,1,1,1,1
5 access_ramospersonalizados_ramo_flor,ramospersonalizados.ramoflor,model_ramospersonalizados_ramoflor,base.group_user,1,1,1,1
6 access_ramospersonalizados_pedido_ramo,ramospersonalizados.pedidoramo,model_ramospersonalizados_pedidoramo,base.group_user,1,1,1,1
7
```

Figura 8: security/ir.model.access.csv

contiene una linea por cada modelo para poner sus permisos



```
<odoo>
  <!-- Vista Formulario para Flores -->
  <record id="view_flor_form" model="ir.ui.view">
    <field name="name">flor.form</field>
    <field name="model">ramospersonalizados.flor</field>
    <field name="arch" type="xml">
      <form string="Flor">
        <sheet>
          <group>
            <field name="name"/>
            <field name="color"/>
            <field name="precio"/>
          </group>
        </sheet>
      </form>
    </field>
  </record>

  <!-- Vista Árbol para Flores -->
  <record id="view_flor_tree" model="ir.ui.view">
    <field name="name">flor.tree</field>
    <field name="model">ramospersonalizados.flor</field>
    <field name="arch" type="xml">
      <tree string="Flores">
        <field name="name"/>
        <field name="color"/>
        <field name="precio"/>
      </tree>
    </field>
  </record>

  <!-- Acción para ver las flores -->
  <record id="action_flor" model="ir.actions.act_window">
    <field name="name">Flores</field>
    <field name="res_model">ramospersonalizados.flor</field>
    <field name="view_mode">tree,form</field>
  </record>

  <!-- Menú Principal -->
  <menuitem id="menu_ramos_personalizados_root" name="Ramos Personalizados" sequence="1"/>

  <!-- Submenú para Flores -->
  <menuitem id="menu_flor" name="Flores" parent="menu_ramos_personalizados_root" action="action_flor"/>
</odoo>
```

Figura 9: views/ViewFlor.xml

view_flor_tree = Vista tipo árbol (lista), donde se mostrarán las flores registradas con sus atributos principales.

view_flor_form = Vista tipo formulario, utilizada para agregar o editar flores individualmente.

Por:David Sánchez Peso

action_flor = Acción que permite abrir la vista de flores en Odoo.

menu_flor = Elemento del menú que permite acceder a la gestión de flores desde el menú principal.

parent = Referencia al menú padre donde se agregará la opción de Flores.

```
<odoo>
<record id="view_pedido_form" model="ir.ui.view">
  <field name="arch" type="xml">
    <form string="Pedido">
      <sheet>
        <field name="name"/>
        <field name="empleado_id"/>
      </group>
      <notebook>
        <page string="Ramos">
          <field name="ramo_ids">
            <tree editable="bottom">
              <field name="ramo_id"/>
              <field name="cantidad"/>
              <field name="precio_total"/>
            </tree>
          </field>
        </page>
      </notebook>
      <group>
        <field name="total"/>
      </group>
    </sheet>
  </form>
</field>
</record>

<!-- Vista Árbol para Pedidos -->
<record id="view_pedido_tree" model="ir.ui.view">
  <field name="name">pedido.tree</field>
  <field name="model">ramospersonalizados.pedido</field>
  <field name="arch" type="xml">
    <tree string="Pedidos">
      <field name="name"/>
      <field name="empleado_id"/>
      <field name="total"/>
    </tree>
  </field>
</record>

<!-- Acción para ver los pedidos -->
<record id="action_pedido" model="ir.actions.act_window">
  <field name="name">Pedidos</field>
  <field name="res_model">ramospersonalizados.pedido</field>
  <field name="view_mode">tree,form</field>
</record>

<!-- Menú Principal -->
<menuitem id="menu_ramos_personalizados_root" name="Ramos Personalizados" sequence="1"/>

<!-- Submenú para Pedidos -->
<menuitem id="menu_pedido" name="Pedidos" parent="menu_ramos_personalizados_root" action="action_pedido"/>
</odoo>
```

Figura 10: views/ViewPedido.xml

Por:David Sánchez Peso

view_pedido_tree = Vista tipo árbol (lista), donde se mostrarán los pedidos registrados con sus atributos principales, como el nombre del pedido, el empleado encargado y el precio total.

view_pedido_form = Vista tipo formulario, utilizada para agregar o editar pedidos individualmente.

action_pedido = Acción que permite abrir la vista de pedidos en Odoo, mostrando tanto la vista de lista como la de formulario.

menu_pedido = Elemento del menú que permite acceder a la gestión de pedidos desde el menú principal.

parent = Referencia al menú padre donde se agregará la opción de Pedidos.

```
<odoo>
<!-- Vista de formulario de Ramo -->
<record id="view_ramo_form" model="ir.ui.view">
  <field name="name">ramo.form</field>
  <field name="model">ramospersonalizados.ramo</field>
  <field name="arch" type="xml">
    <form string="Ramo">
      <sheet>
        <group>
          <field name="name"/>
          <field name="precio"/>
          <field name="flores_ids">
            <tree editable="bottom">
              <field name="flor_id"/>
              <field name="cantidad"/>
              <field name="precio_total" readonly="1"/>
            </tree>
          </field>
        </group>
      </sheet>
    </form>
  </field>
</record>

<!-- Vista de lista de Ramos -->
<record id="view_ramo_tree" model="ir.ui.view">
  <field name="name">ramo.tree</field>
  <field name="model">ramospersonalizados.ramo</field>
  <field name="arch" type="xml">
    <tree string="Ramos">
      <field name="name"/>
      <field name="precio"/>
    </tree>
  </field>
</record>

<!-- Acción de la vista de Ramos -->
<record id="action_ramo" model="ir.actions.act_window">
  <field name="name">Ramos</field>
  <field name="res_model">ramospersonalizados.ramo</field>
  <field name="view_mode">tree,form</field>
</record>

<!-- Submenú para Ramos -->
<menuitem id="menu_ramo" name="Ramos" parent="menu_ramos_personalizados_root" action="action_ramo"/>
</odoo>
```

Figura 11: views/ViewRamo.xml

view_ramo_tree = Vista tipo árbol (lista), donde se mostrarán los ramos registrados con sus atributos principales, como el nombre del ramo, el precio total y las flores que lo componen.
view_ramo_form = Vista tipo formulario, utilizada para agregar o editar ramos individualmente, permitiendo seleccionar las flores y su cantidad.

Por:David Sánchez Peso

action_ramo = Acción que permite abrir la vista de ramos en Odoo, mostrando tanto la vista de lista como la de formulario.

menu_ramo = Elemento del menú que permite acceder a la gestión de ramos desde el menú principal.

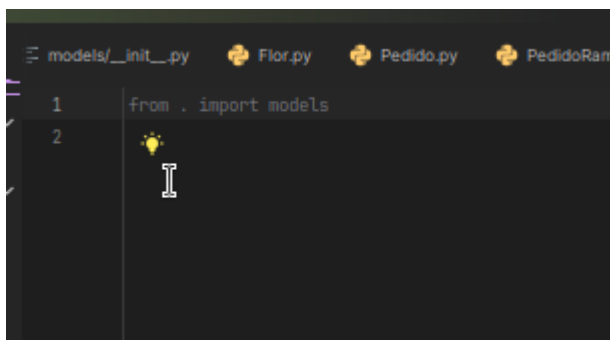


Figura 12: __init__.py

importa los modelos de la carpeta models

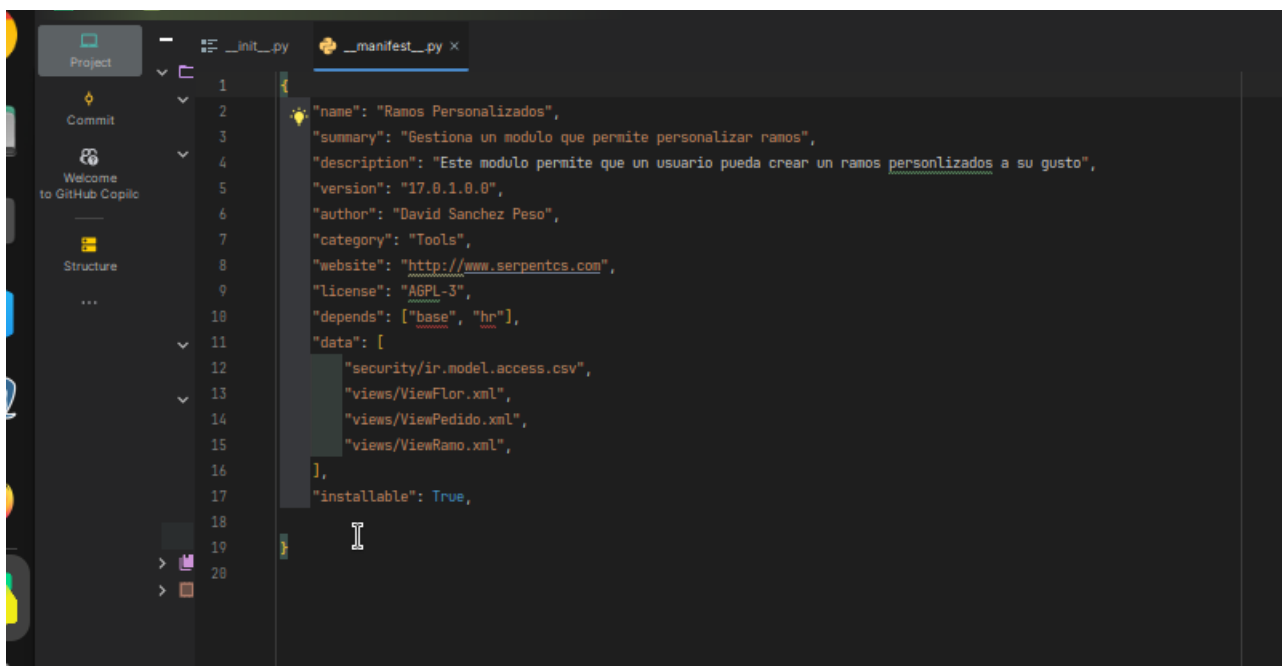


Figura 13: __manifest__.py

4. Implementación de dos modelos (1 punto)

El módulo cuenta con dos modelos principales: **Flor** y **Ramo**, que están relacionados entre sí. El modelo **Flor** representa cada tipo de flor disponible, con atributos como **nombre**, **color** y **precio** (que no puede ser negativo). Estas flores se usan para armar ramos personalizados. Por otro lado, el modelo **Ramo** agrupa varias flores en distintas cantidades para formar un producto final, permitiendo asignarle un **nombre** y calculando su **precio total** automáticamente según las flores que contiene. La relación entre ambos es de **uno a muchos**, ya que un ramo puede incluir varias

Por:David Sánchez Peso

flores, pero una misma flor puede estar en distintos ramos. Esto se gestiona con una tabla intermedia que asocia cada ramo con las flores que lo componen y sus cantidades.

5. Creación de más modelos (1 punto)

Hay otro modelo más llamado Pedido, que está relacionado con Ramo a través de una tabla intermedia. En este modelo, se registran los pedidos de los clientes, especificando qué ramos están incluidos en cada uno, junto con la cantidad de cada ramo y el empleado responsable del pedido. El modelo Ramo define los ramos disponibles, con su nombre, precio total calculado a partir de las flores que lo componen, y la relación con las flores a través de una tabla intermedia que gestiona las cantidades de cada flor en el ramo. La tabla intermedia entre Pedido y Ramo permite gestionar los ramos específicos que forman parte de cada pedido.

6. Creación de las vistas, modelos y menús dentro de Odoo (0.5 punto)

Nombre del ramo

Flores	Flor	Canti...	Precio T...
	Flor	1	1,00

Agregar una línea

Figura 14: Ramos dentro de odoo

Por:David Sánchez Peso

Ramos Personalizados Flores Pedidos Ramos

Nuevo Pedidos Nuevo

Nombre del Pedido pedido1

Empleado Responsable ABRAHAM IZQUIERDO PUIQ

Ramos

Ramo	Canti...	Precio T...
ramo3	2	68,00

Agregar una línea

Total 68,00

Figura 15: Pedidos dentro de odoo

Nuevo Flores Nuevo

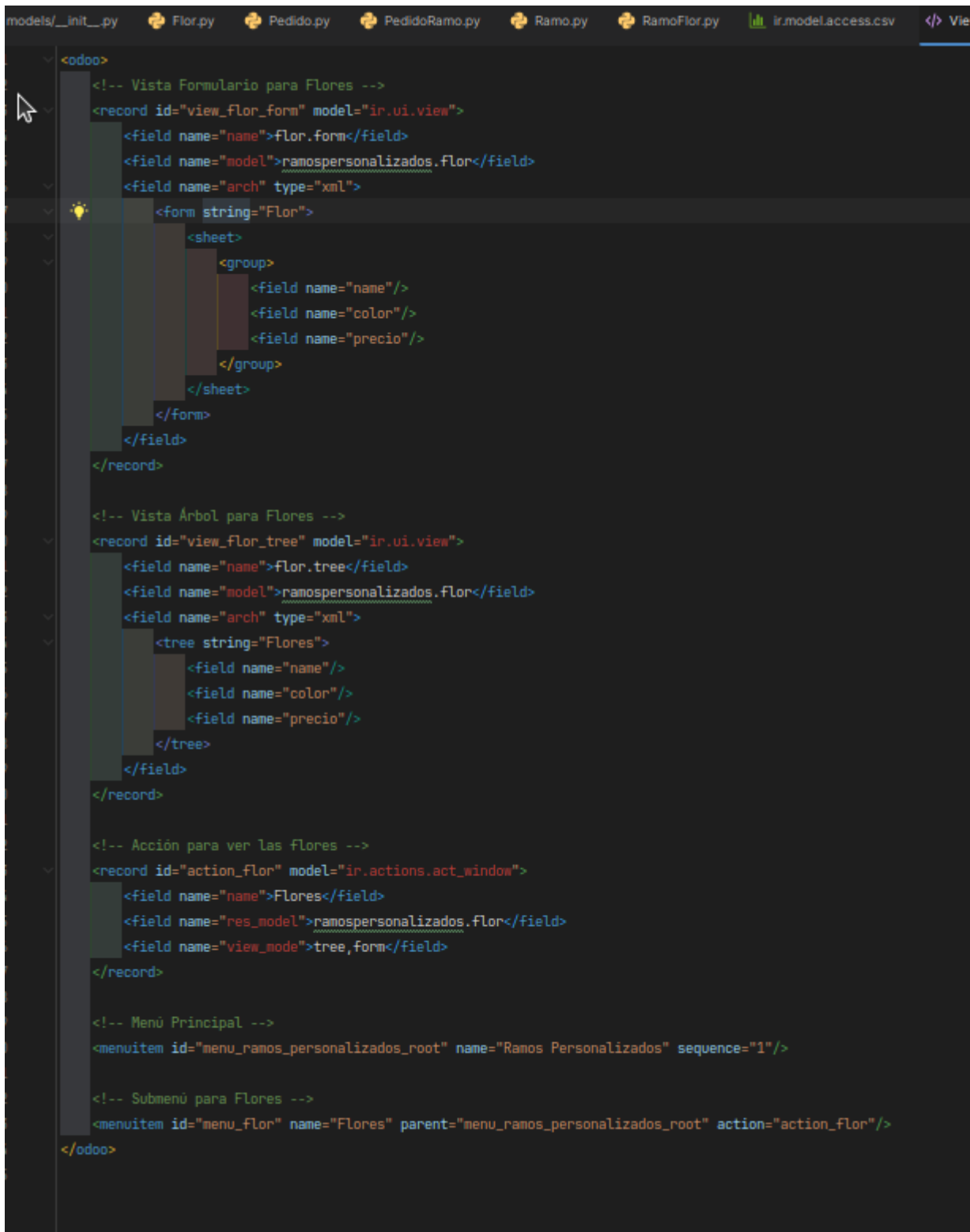
Nombre de la flor b

Color verde

pgAdmin 4 idad 1,00

Figura 16: Flores dentro de odoo

7. Modificación de las vistas (1.5 punto)

The image shows a code editor with several Python files open at the top: models/__init__.py, Flor.py, Pedido.py, PedidoRamo.py, Ramo.py, RamoFlor.py, and ir.model.access.csv. The active file is ViewFlor.xml, which contains Odoo XML code. The code defines three views: a form view (view_flor_form), a tree view (view_flor_tree), and an action (action_flor). The form view includes fields for name, color, and price, grouped under a sheet. The tree view includes fields for name, color, and price. The action view is for viewing flowers. The code also includes menu items for the main menu and a submenu for flowers.

```
<odoo>
<!-- Vista Formulario para Flores -->
<record id="view_flor_form" model="ir.ui.view">
  <field name="name">Flor.form</field>
  <field name="model">ramospersonalizados.flor</field>
  <field name="arch" type="xml">
    <form string="Flor">
      <sheet>
        <group>
          <field name="name"/>
          <field name="color"/>
          <field name="precio"/>
        </group>
      </sheet>
    </form>
  </field>
</record>

<!-- Vista Árbol para Flores -->
<record id="view_flor_tree" model="ir.ui.view">
  <field name="name">Flor.tree</field>
  <field name="model">ramospersonalizados.flor</field>
  <field name="arch" type="xml">
    <tree string="Flores">
      <field name="name"/>
      <field name="color"/>
      <field name="precio"/>
    </tree>
  </field>
</record>

<!-- Acción para ver las flores -->
<record id="action_flor" model="ir.actions.act_window">
  <field name="name">Flores</field>
  <field name="res_model">ramospersonalizados.flor</field>
  <field name="view_mode">tree,form</field>
</record>

<!-- Menú Principal -->
<menuitem id="menu_ramos_personalizados_root" name="Ramos Personalizados" sequence="1"/>

<!-- Submenú para Flores -->
<menuitem id="menu_flor" name="Flores" parent="menu_ramos_personalizados_root" action="action_flor"/>
</odoo>
```

Figura 17: Código de ViewFlor

Se ha creado una vista tipo árbol para mostrar las flores disponibles con sus atributos principales. Se ha creado una vista tipo formulario para agregar y editar flores, con un campo de precio que

Por:David Sánchez Peso

debe ser positivo, gracias a la restricción que hemos implementado.

Se ha añadido un widget numérico para que el precio de la flor se ingrese de manera correcta y fácil.

Vista Tipo Árbol (tree):

Muestra las flores con nombre, color y precio.

Vista Tipo Formulario (form):

Permite agregar y editar flores, asegurando que el campo precio sea numérico y no vacío.

Widget float:

Se ha añadido un widget de tipo numérico (float) en el campo precio, para asegurarse de que el valor ingresado sea correcto y con formato adecuado.

```
<odoo>
<record id="view_pedido_form" model="ir.ui.view">
  <field name="arch" type="xml">
    <form string="Pedido">
      <sheet>
        <field name="name"/>
        <field name="empleado_id"/>
      </group>
      <notebook>
        <page string="Ramos">
          <field name="ramo_ids">
            <tree editable="bottom">
              <field name="ramo_id"/>
              <field name="cantidad"/>
              <field name="precio_total"/>
            </tree>
          </field>
        </page>
      </notebook>
      <group>
        <field name="total"/>
      </group>
    </sheet>
  </form>
</field>
</record>

<!-- Vista Árbol para Pedidos -->
<record id="view_pedido_tree" model="ir.ui.view">
  <field name="name">pedido.tree</field>
  <field name="model">ramospersonalizados.pedido</field>
  <field name="arch" type="xml">
    <tree string="Pedidos">
      <field name="name"/>
      <field name="empleado_id"/>
      <field name="total"/>
    </tree>
  </field>
</record>

<!-- Acción para ver los pedidos -->
<record id="action_pedido" model="ir.actions.act_window">
  <field name="name">Pedidos</field>
  <field name="res_model">ramospersonalizados.pedido</field>
  <field name="view_mode">tree,form</field>
</record>

<!-- Menú Principal -->
<menuitem id="menu_ramos_personalizados_root" name="Ramos Personalizados" sequence="1"/>

<!-- Submenú para Pedidos -->
<menuitem id="menu_pedido" name="Pedidos" parent="menu_ramos_personalizados_root" action="action_pedido"/>
</odoo>
```

Figura 18: código de ViewPedido

Por:David Sánchez Peso

Se ha creado una vista tipo árbol para mostrar los pedidos registrados, que incluye el nombre del pedido, el empleado encargado y el precio total.

Se ha añadido una vista tipo formulario para agregar o editar pedidos, donde se pueden seleccionar los ramos y cantidades.

Se ha agregado un widget de selección de ramos para facilitar la elección de ramos en la vista de formulario.

Vista Tipo Árbol (tree):

Muestra los pedidos con el nombre, empleado asignado y precio total.

Vista Tipo Formulario (form):

Permite la creación y edición de pedidos, con un campo para seleccionar los ramos y ver el precio total.

Widget many2many_tags:

Se añadió el widget many2many_tags al campo ramos_ids para mostrar los ramos seleccionados como etiquetas de manera más amigable en el formulario.

```
<odoo>
<!-- Vista de formulario de Ramo -->
<record id="view_ramo_form" model="ir.ui.view">
  <field name="name">ramo.form</field>
  <field name="model">ramospersonalizados.ramo</field>
  <field name="arch" type="xml">
    <form string="Ramo">
      <sheet>
        <group>
          <field name="name"/>
          <field name="precio"/>
          <field name="flores_ids">
            <tree editable="bottom">
              <field name="flor_id"/>
              <field name="cantidad"/>
              <field name="precio_total" readonly="1"/>
            </tree>
          </field>
        </group>
      </sheet>
    </form>
  </field>
</record>

<!-- Vista de lista de Ramos -->
<record id="view_ramo_tree" model="ir.ui.view">
  <field name="name">ramo.tree</field>
  <field name="model">ramospersonalizados.ramo</field>
  <field name="arch" type="xml">
    <tree string="Ramos">
      <field name="name"/>
      <field name="precio"/>
    </tree>
  </field>
</record>

<!-- Acción de la vista de Ramos -->
<record id="action_ramo" model="ir.actions.act_window">
  <field name="name">Ramos</field>
  <field name="res_model">ramospersonalizados.ramo</field>
  <field name="view_mode">tree,form</field>
</record>

<!-- Submenú para Ramos -->
<menuitem id="menu_ramo" name="Ramos" parent="menu_ramos_personalizados_root" action="action_ramo"/>
</odoo>
```

Figura 19: Código de ViewRamo

Se ha creado una vista tipo árbol para listar los ramos registrados, mostrando su nombre, precio total y las flores asociadas.

Se ha modificado la vista tipo formulario para que permita la selección de flores y cantidades.

Por:David Sánchez Peso

Se ha añadido un widget de cálculo de precio que permite calcular automáticamente el precio total del ramo en la vista de formulario.

Vista Tipo Árbol (tree):

Muestra los ramos con su nombre, precio y las flores asociadas.

Vista Tipo Formulario (form):

Permite editar ramos, con un campo de flores que se seleccionan usando many2many_tags para mostrar etiquetas.

Campo precio en Solo Lectura:

El precio se calcula automáticamente, pero se muestra en solo lectura para evitar que se modifique manualmente.

8. Restricciones en los campos (1.5 punto)

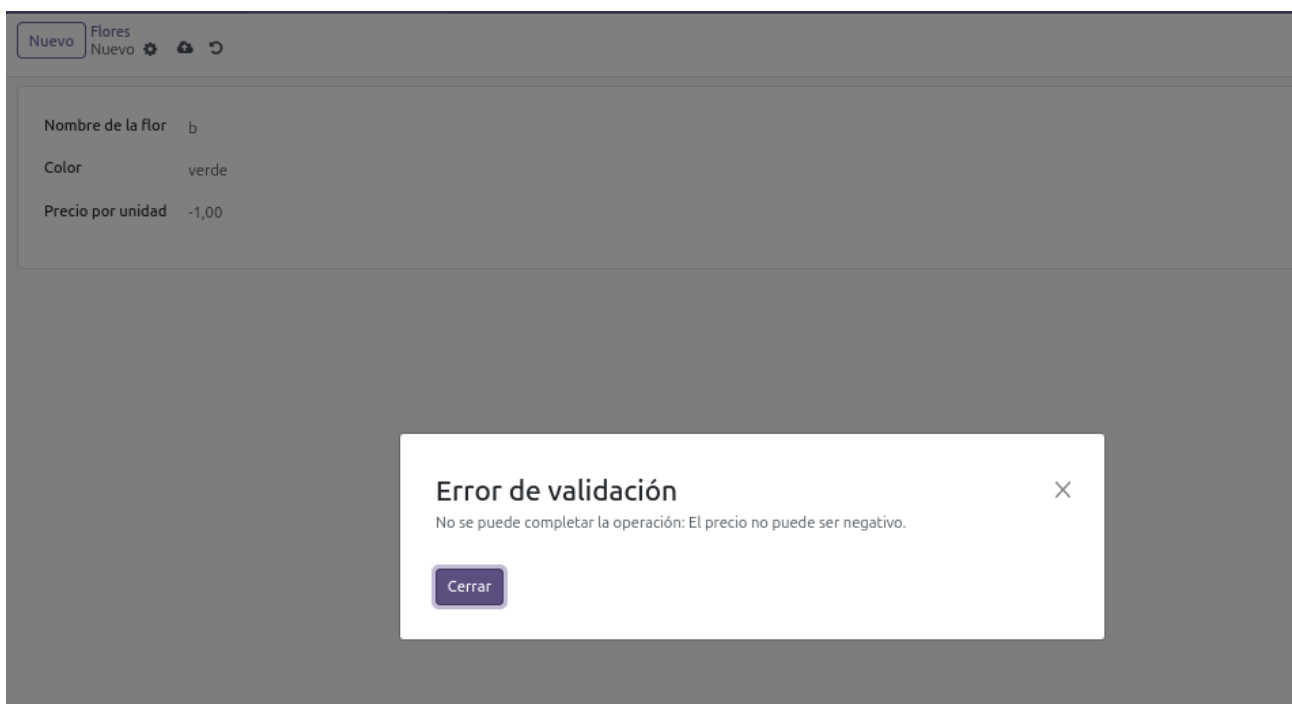
The screenshot shows a web application interface for managing flowers. At the top, there's a header with a 'Nuevo' button and a 'Flores Nuevo' link. Below this, there's a form with three fields: 'Nombre de la flor' with the value 'b', 'Color' with the value 'verde', and 'Precio por unidad' with the value '-1,00'. A modal dialog box is displayed in the center, titled 'Error de validación'. The message inside says 'No se puede completar la operación: El precio no puede ser negativo.' and there is a 'Cerrar' button at the bottom.

Figura 20: Muestra de que la restriccion funciona

```
name = fields.Char(string="Nombre de la flor", required=True)
color = fields.Char(string="Color")
precio = fields.Float(string="Precio por unidad", required=True) # Precio de cada flor

_sql_constraints = [
    ('check_precio_no_negativo', 'CHECK(precio >= 0)', 'El precio no puede ser negativo.')
]
```

Figura 21: Código de flor para ver la restriccion

Por:David Sánchez Peso

Cuando un usuario intenta guardar una flor con un precio negativo, Odoo muestra un mensaje de error y no permite guardar el registro. Este comportamiento se puede observar directamente en la interfaz de usuario, donde se previene la entrada de valores incorrectos.

9. pgAdmin (0.5 punto)

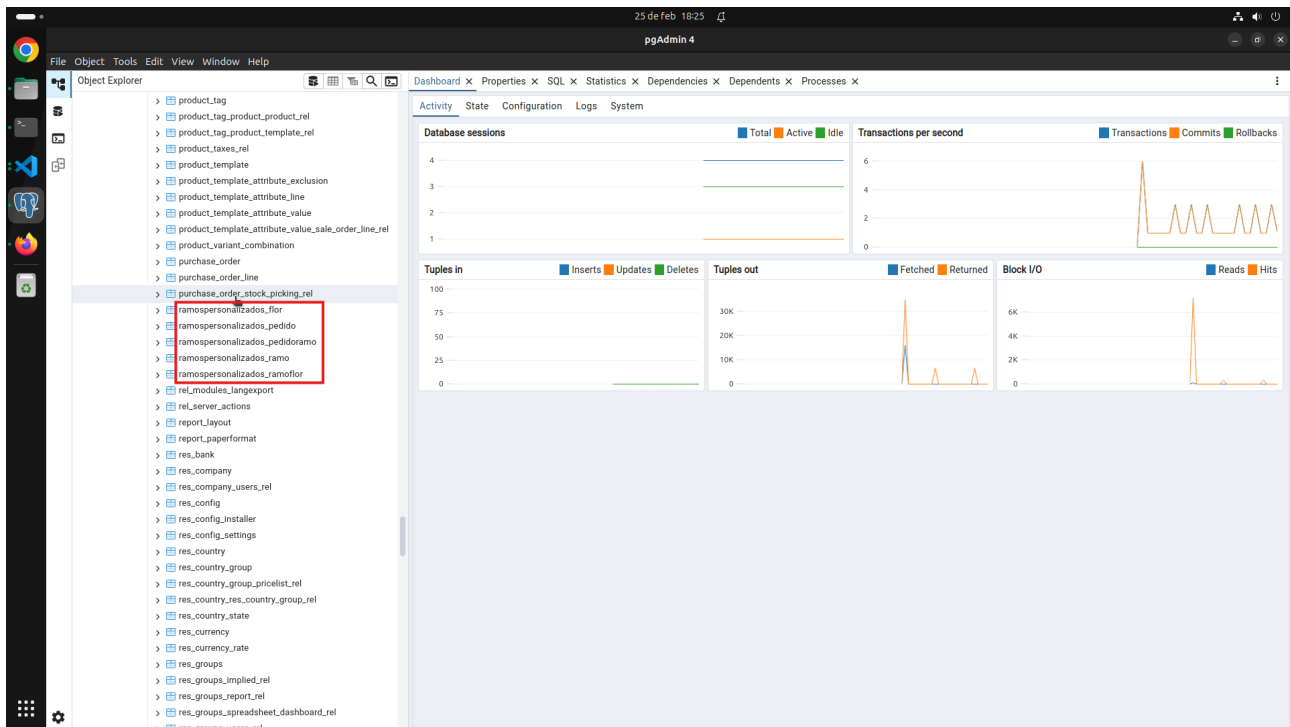


Figura 22: Imagen de pgadmin de las tablas creadas

Por:David Sánchez Peso