

0. Escenario

En tareas previas hemos creado un módulo autónomo que, al instalarlo, creaba su propia tabla en la base de datos y un nuevo menú con vistas formulario y lista.

En esta tarea vamos a modificar un módulo que ya existe y vamos a programar alguna pequeña utilidad. En este caso modificaremos el módulo de contactos para añadir un nuevo campo, la fecha de nacimiento. Queremos darle alguna utilidad al módulo, no llega con que simplemente almacene la información que proporciona el usuario así que vamos a introducir código para que calcule la edad y otro para que calcule el signo del zodiaco. Piensa que una vez que sepas donde y como insertar código ya podrás hacer lo que quieras. También vamos a modificar algunas vistas para incluir esta nueva información de manera gráfica.

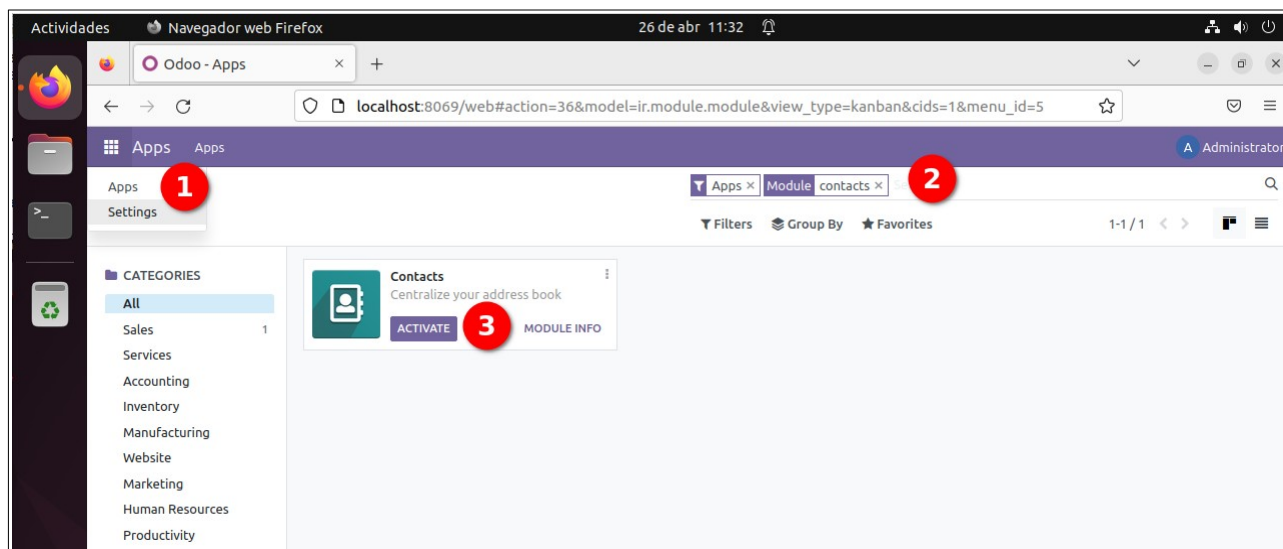
La idea es que conozcas las posibilidades infinitas de ampliación que tiene Odoo ya que te permite programar algo totalmente nuevo o bien ampliar/modificar algo que ya existe

1. Preparación de la máquina virtual

Partiremos de un Odoo recién instalado como servicio.

Como se puede ver en la captura el Odoo del que parto no tiene nada instalado mas allá del módulo base obligatorio.

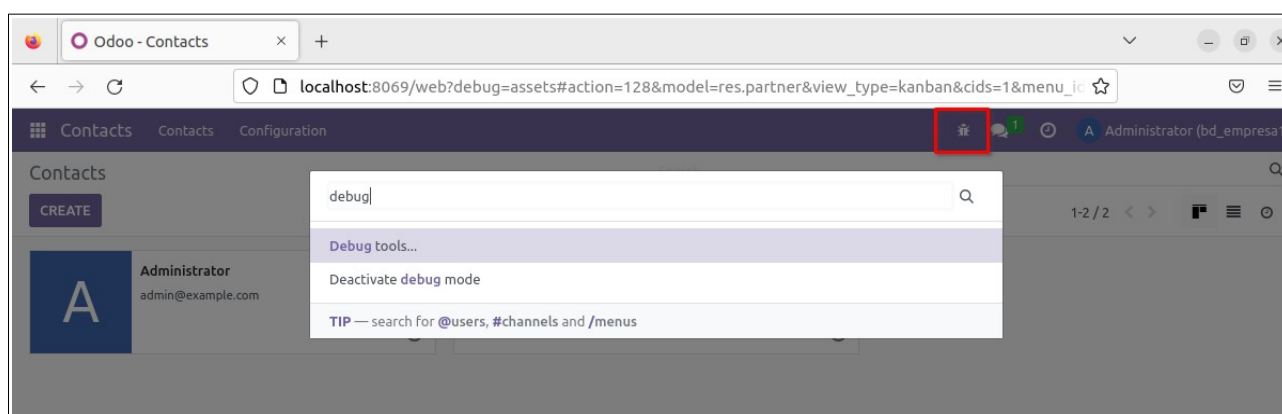
La tarea consiste en modificar el módulo de contactos para dotarlo de nueva funcionalidad añadida así que vamos a instalarlo. Para ello pulsaremos en el menú de la esquina superior izquierda y seleccionamos la entrada de "aplicaciones". En esta ventana hacemos uso del buscador para localizar la aplicación de "Contactos" y la instalamos pulsando en "activar"



Odoo instalará el módulo leyendo su archivo de modelo en python para crear las tablas en la base de datos y demás funcionalidad programada. También añadirá las vistas y los menús a la interfaz gráfica de Odoo. Si todo termina sin errores se recargará la página y ya tendremos acceso a la aplicación de contactos desde el menú de la esquina superior izquierda.

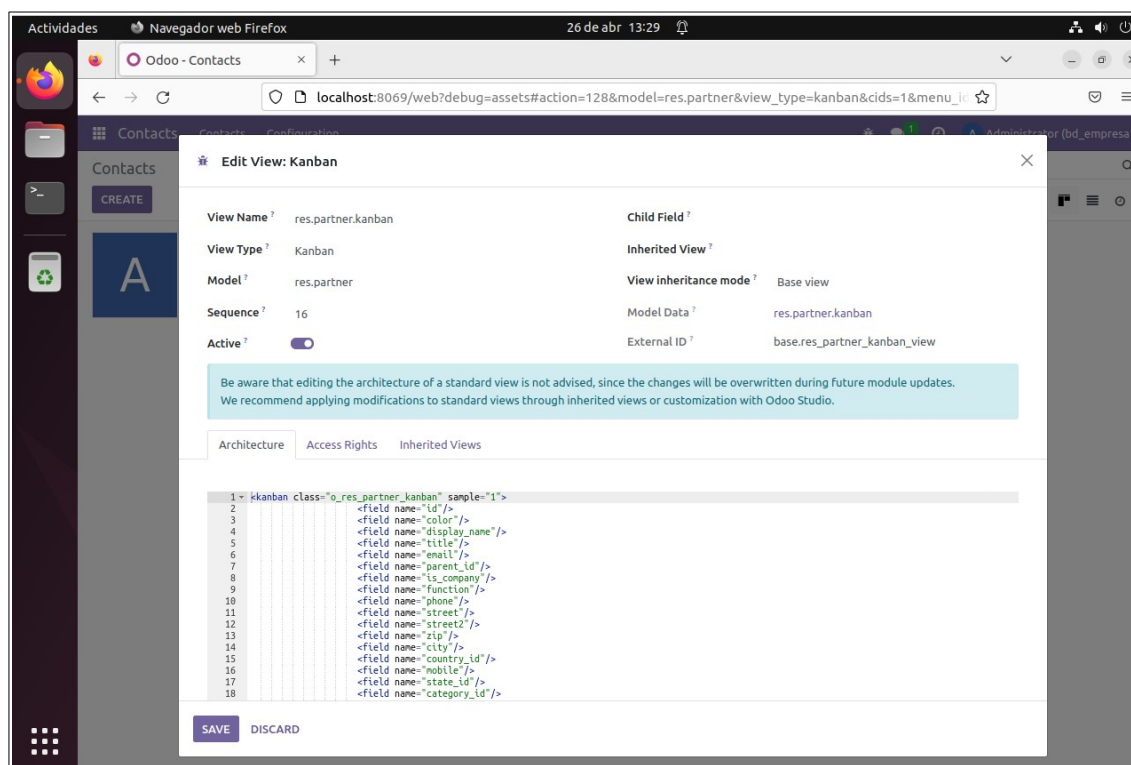
Al entrar en la aplicación de contactos se muestra por defecto la vista kanban en la que los registros se muestran de manera resumida en forma de etiquetas que rellenan un mosaico. En la selección de vistas (un menú que tenemos en la zona superior derecha) también tenemos disponible una vista en modo lista donde los registros se ven en forma de tabla. En este caso los registros están resumidos pero con algo más de información que en la vista kanban.

Si pinchamos en cualquiera de los registros disponibles entramos en la vista formulario donde solo veremos un registro pero con toda la información.



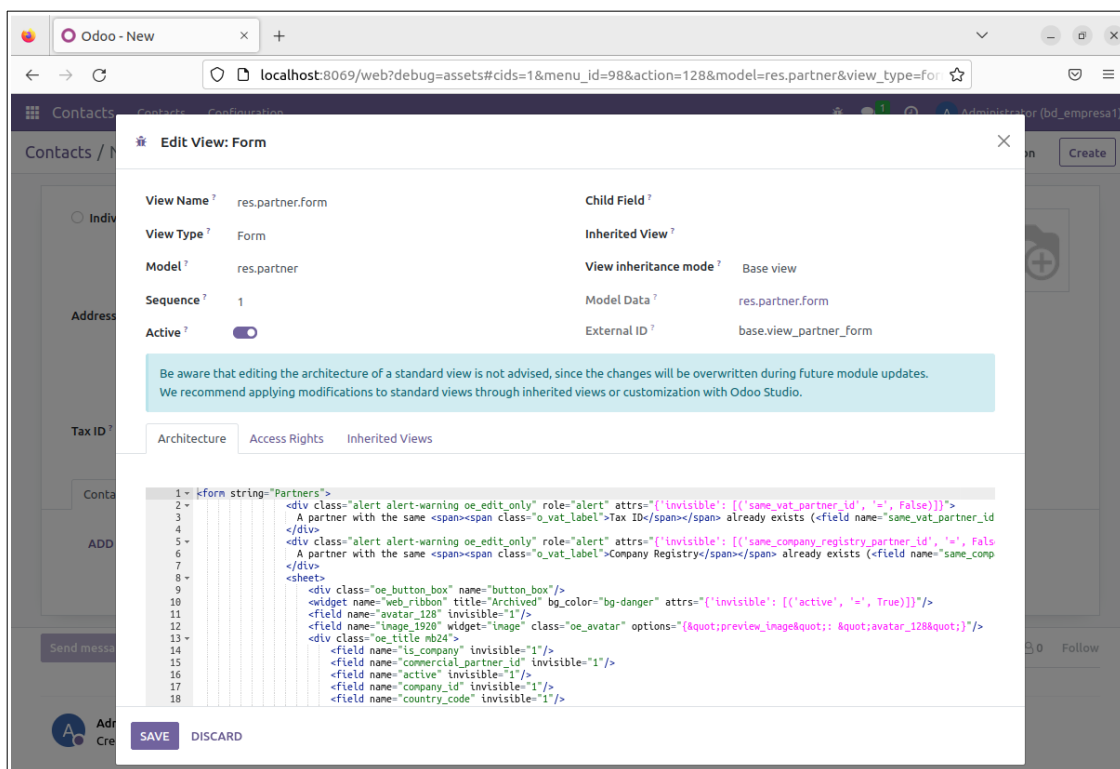
Con la combinación de teclas Ctrl+K podemos buscar y activar el "modo debug" y así poder ver más información técnica de Odoo. Tienes el modo debug activado si puedes ver un icono en forma de bicho en la parte derecha del menú superior.

En ese icono tenemos la opción de edit view:kanban en la que podemos ver el código XML y otros datos de la vista como el nombre interno o el modelo al que está asociado.



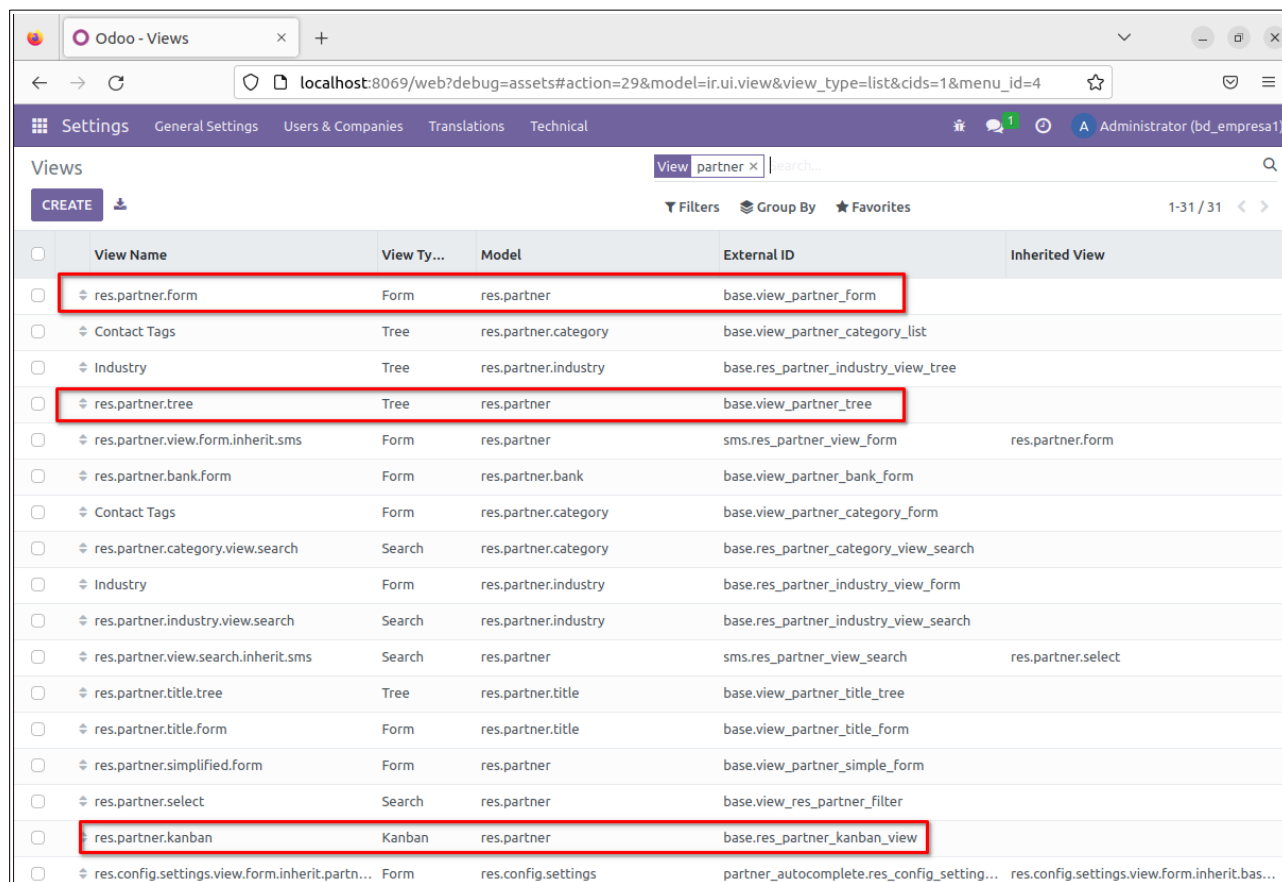
En este caso vemos que esta vista se llama res.partner.kanban y que está asociada al model res.partner.

Si pinchamos en crear un nuevo contacto se abrirá una vista tipo formulario. Repitiendo el proceso anterior tenemos acceso a edit view: form.



Aquí podemos ver que esta vista se llama `res.partner.form` y también representa al model `res.partner`.

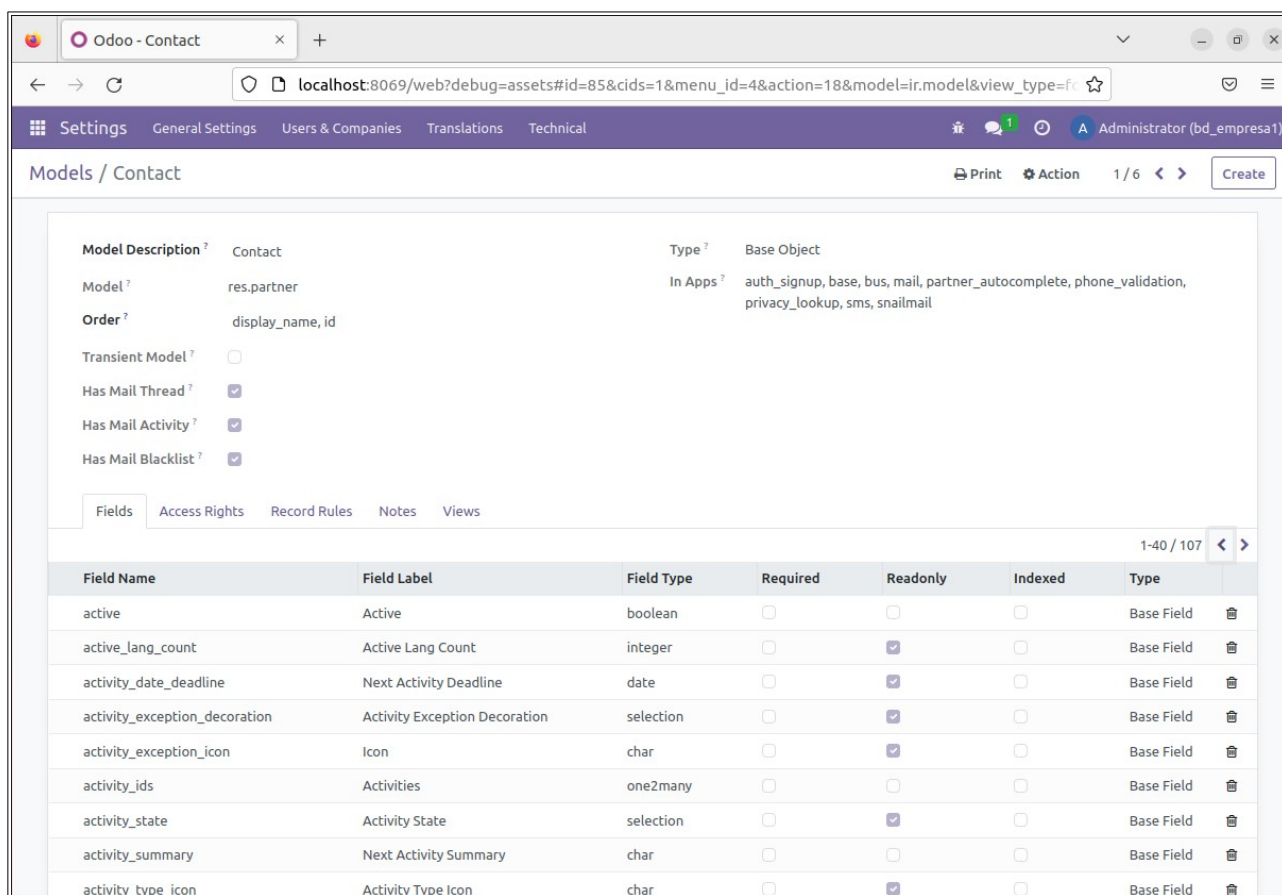
Desde el menú Ajustes / Técnico / Vistas podremos localizar las vistas que nos interesan



| | View Name | View Ty... | Model | External ID | Inherited View |
|--------------------------|--|------------|----------------------|--|--|
| <input type="checkbox"/> | res.partner.form | Form | res.partner | base.view_partner_form | |
| <input type="checkbox"/> | Contact Tags | Tree | res.partner.category | base.view_partner_category_list | |
| <input type="checkbox"/> | Industry | Tree | res.partner.industry | base.res_partner_industry_view_tree | |
| <input type="checkbox"/> | res.partner.tree | Tree | res.partner | base.view_partner_tree | |
| <input type="checkbox"/> | res.partner.view.form.inherit.sms | Form | res.partner | sms.res_partner_view_form | res.partner.form |
| <input type="checkbox"/> | res.partner.bank.form | Form | res.partner.bank | base.view_partner_bank_form | |
| <input type="checkbox"/> | Contact Tags | Form | res.partner.category | base.view_partner_category_form | |
| <input type="checkbox"/> | res.partner.category.view.search | Search | res.partner.category | base.res_partner_category_view_search | |
| <input type="checkbox"/> | Industry | Form | res.partner.industry | base.res_partner_industry_view_form | |
| <input type="checkbox"/> | res.partner.industry.view.search | Search | res.partner.industry | base.res_partner_industry_view_search | |
| <input type="checkbox"/> | res.partner.view.search.inherit.sms | Search | res.partner | sms.res_partner_view_search | res.partner.select |
| <input type="checkbox"/> | res.partner.title.tree | Tree | res.partner.title | base.view_partner_title_tree | |
| <input type="checkbox"/> | res.partner.title.form | Form | res.partner.title | base.view_partner_title_form | |
| <input type="checkbox"/> | res.partner.simplified.form | Form | res.partner | base.view_partner_simple_form | |
| <input type="checkbox"/> | res.partner.select | Search | res.partner | base.view_res_partner_filter | |
| <input type="checkbox"/> | res.partner.kanban | Kanban | res.partner | base.res_partner_kanban_view | |
| <input type="checkbox"/> | res.config.settings.view.form.inherit.partn... | Form | res.config.settings | partner_autocomplete.res_config_setting... | res.config.settings.view.form.inherit.bas... |

Desde el menú Ajustes / Técnico / Modelos localizaremos el modelo `res.partner`. En este modelo añadiremos los nuevos campos que queremos incluir pero desde aquí podemos ver los campos que ya tiene este modelo y las vistas que muestran sus datos.

Si te fijas en los campos verás que no se guarda la fecha de nacimiento ni la edad y mucho menos el signo del zodiaco.



Model Description ? Contact

Model ? res.partner

Order ? display_name, id

Transient Model ? ☐

Has Mail Thread ? ☒

Has Mail Activity ? ☒

Has Mail Blacklist ? ☒

Type ? Base Object

In Apps ? auth_signup, base, bus, mail, partner_autocomplete, phone_validation, privacy_lookup, sms, snailmail

Fields | Access Rights | Record Rules | Notes | Views

1-40 / 107

| Field Name | Field Label | Field Type | Required | Readonly | Indexed | Type |
|-------------------------------|-------------------------------|------------|--------------------------|-------------------------------------|--------------------------|------------|
| active | Active | boolean | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | Base Field |
| active_lang_count | Active Lang Count | integer | <input type="checkbox"/> | <input checked="" type="checkbox"/> | <input type="checkbox"/> | Base Field |
| activity_date_deadline | Next Activity Deadline | date | <input type="checkbox"/> | <input checked="" type="checkbox"/> | <input type="checkbox"/> | Base Field |
| activity_exception_decoration | Activity Exception Decoration | selection | <input type="checkbox"/> | <input checked="" type="checkbox"/> | <input type="checkbox"/> | Base Field |
| activity_exception_icon | Icon | char | <input type="checkbox"/> | <input checked="" type="checkbox"/> | <input type="checkbox"/> | Base Field |
| activity_ids | Activities | one2many | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | Base Field |
| activity_state | Activity State | selection | <input type="checkbox"/> | <input checked="" type="checkbox"/> | <input type="checkbox"/> | Base Field |
| activity_summary | Next Activity Summary | char | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | Base Field |
| activity_type_icon | Activity Type Icon | char | <input type="checkbox"/> | <input checked="" type="checkbox"/> | <input type="checkbox"/> | Base Field |

2. Creando el módulo

En este apartado me remito a la guía de la tarea UD3 donde se creaba un módulo para Odoo. Recuerda que es buena costumbre tener una carpeta dedicada a nuestros propios módulos de manera que si alguna vez tenemos que reinstalar Odoo no se sobrescriban nuestros módulos.

Para esto debíamos crear una carpeta donde almacenar nuestros módulos y después añadir su ruta al archivo de configuración de Odoo. Reiniciamos el servicio para que acepte el nuevo cambio y listo.

El siguiente paso consiste en crear la estructura del módulo con todas sus subcarpetas, archivos y plantillas por defecto. Odoo proporciona una manera sencilla y automatizada de hacer esto mediante el comando:

```
opt/odoo/odoo-bin scaffold horoscopo /opt/odoo/modulos_extra
```

El primer paso opcional consiste en modificar el archivo `__manifest__.py` en la raíz de la carpeta de nuestro módulo.

Ahora vamos a reiniciar el servicio mediante el siguiente comando:

```
sudo service odoo16 restart
```

Con esto Odoo ya debería ser capaz de presentar este módulo pero es posible que tengas que refrescar la lista de aplicaciones. Usa el buscador para localizar el módulo "Horóscopo" que acabamos de crear pero recuerda quitar el filtro de aplicaciones (horóscopo es un módulo)

3. Creando el modelo

En la carpeta models del módulo horóscopo encontraremos un archivo python llamado models.py que será donde indiquemos a Odoo que queremos añadir unos campos al modelo ya existente de res.partner (el usado en contactos) y el código para realizar los cálculos.

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

from odoo import models, fields, api

class horoscopo(models.Model):
    _inherit = "res.partner"

    cumple = fields.Date("Cumpleaños")
    edad = fields.Integer(string="Edad", readonly=True,
compute="_calcula_edad", store=True)
    signo = fields.Char(string="Signo zodiaco", readonly=True,
compute="_calcula_signo", store=True)

    @api.depends("cumple")
    def _calcula_edad(self):
        for registro in self:
            edad = 99
            registro.edad = edad

    @api.depends("cumple")
    def _calcula_signo(self):
        self.ensure_one()
        signo_zodiaco = "Mi signo"
        self.signo = signo_zodiaco
```

En este código lo primero que hacemos es indicar que esta clase hereda del modelo res.partner que es el que gestiona los contactos.

Al no ponerle el atributo _name estamos haciendo que añada campos en el modelo res.partner en lugar de generar una nueva tabla en la base de datos para este modelo zodiaco.

Las siguientes líneas son los nuevos campos que añadimos. El primero es la propia fecha de nacimiento definida con su tipo adecuado, Date.

Las dos siguientes representan campos calculados como la edad y el signo del zodiaco. Ambas tienen modificadores que indican que queremos que sean campos de solo lectura, que se quiere guardar el resultado en la tabla (para que esté accesible a otros módulos) y otro modificador donde se indica la función python que se debe que ejecutar para calcular el campo.

El resto del código son las dos funciones que usaremos para calcular cada uno de los campos. Como ves son funciones sencillas definidas en Python donde tendrá que completar el código para que funcionen.

Como ves ambas tienen el decorador `@api.depends("cumple")` que indica que la función se ejecutará cada vez que el valor del campo "cumple" cambie y que para realizar el cálculo se usarán valores de otros campos.

Cuando al cambiar el valor del campo "cumple" se desencadene la ejecución de la función indicada, Odoo pasará a la función, a través de self un recordset compuesto de todos los registros cuyo campo "cumple" haya cambiado. Este cambio se puede deber a la interacción del usuario o por la acción programada de cualquier otro módulo. Por ello es habitual recorrer este self por si varios registros han cambiado.

La función que calcula el signo del zodiaco en función de la fecha de nacimiento tiene la misma filosofía que la anterior, casi se podría copiar y pegar pero en este caso, el primer comando fuerza a que self (que originalmente tenía un recordset) se quede solo con un registro. De esta manera ya no hace falta recorrer con un for y podemos escribir directamente en el self. En caso que el recordset tuviera más de un elemento generaría una excepción que podríamos capturar y tratar.

A modo de resumen, si estamos programando una función que queremos que se ejecute cuando detectemos un cambio que puede venir por interacción del usuarios, de sensores, de internet,... entonces debemos elegir la primera opción ya que es posible y probable que cambien varios registros a la vez y, por tanto que se actualicen todos de manera automática

Si solo queremos que el usuario modifique un registro y ningún otro se va a ver afectado, entonces la arquitectura de la segunda función es más adecuada.

Si solo queremos modificar un registro pero también queremos ser conscientes de si otros han cambiado entonces debemos usar esta segunda opción pero añadiendo código que capture la excepción.

Hay que tener en cuenta que la primera vez que instalemos el módulo todos los campos "cumple" van a cambiar solo por el hecho de crearlos y por lo tanto el recordset estará compuesto por todos los registros que tenga contactos en ese momento. Por ello es necesario no solo pensar en el día a día del módulo sino también pensar en el momento de su integración. Tal como está ahora mismo genera un error cuando calcula el signo puesto que espera recibir un solo registro y está recibiendo todos.

Odoo Error

```
File "/opt/odoo/odoo/modules/registry.py", line 465, in init_models
    func()
File "/opt/odoo/odoo/addons/base/models/ir_model.py", line 1720, in _reflect_relation
    self.env.invalidate_all()
File "/opt/odoo/odoo/api.py", line 722, in invalidate_all
    self.flush_all()
File "/opt/odoo/odoo/api.py", line 732, in flush_all
    self._recompute_all()
File "/opt/odoo/odoo/api.py", line 728, in _recompute_all
    self[field.model_name]._recompute_field(field)
File "/opt/odoo/odoo/models.py", line 6163, in _recompute_field
    field.recompute(records)
File "/opt/odoo/odoo/fields.py", line 1324, in recompute
    self.compute_value(recs)
File "/opt/odoo/odoo/fields.py", line 1346, in compute_value
    records._compute_field_value(self)
File "/opt/odoo/addons/mail/models/mail_thread.py", line 403, in _compute_field_value
    return super()._compute_field_value(field)
File "/opt/odoo/odoo/models.py", line 4196, in _compute_field_value
    getattr(self, field.compute)()
File "/opt/odoo/modulos_extra/horoscopo/models/models.py", line 20, in _calcula_signo
    self.ensure_one()
File "/opt/odoo/odoo/models.py", line 5115, in ensure_one
    raise ValueError("Expected singleton: %s" % self)
ValueError: Expected singleton: res.partner(5, 4, 6, 2, 7, 8, 1, 3)

The above server error caused the following client error:
RPC_ERROR: Odoo Server Error
RPCError@http://localhost:8069/web/assets/debug/web.assets_backend.js:10126:9 (/web/static/src/core/network/rpc_service.js:11)
makeErrorFromResponse@http://localhost:8069/web/assets/debug/web.assets_backend.js:10149:19 (/web/static/src/core/network/rpc_serv:
jsonrpc/promise</>@http://localhost:8069/web/assets/debug/web.assets_backend.js:10202:48 (/web/static/src/core/network/rpc_service.
```

OK

Para que todo funcione en cualquier situación debemos modificar el código quedando así:

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

from odoo import models, fields, api

class horoscopo(models.Model):
    _inherit = "res.partner"
    cumple = fields.Date("Cumpleaños")
    edad = fields.Integer(string="Edad", readonly=True,
compute="_calcula_edad", store=True)
    signo = fields.Char(string="Signo zodiaco", readonly=True,
compute="_calcula_signo", store=True)

@api.depends("cumple")
def _calcula_edad(self):
    for registro in self:
        if registro.cumple:
            #Inserta aquí el código para calcular la edad
            edad=99
            registro.edad = edad

@api.depends("cumple")
```



```
def _calcula_signo(self):
    try:
        self.ensure_one()
        #Inserta aquí el código para calcular el signo del zodiaco
        self.signo = "Sin signo"
    except Exception:
        print("Varios registros en el dataset de _calcula_signo")
```

En este código modificado los cambios en la primera función hace que recorra todo el dataset y que solo en caso de tener información en el campo "cumple" aplique el cálculo. En la primera ejecución cuando se instale el módulo el campo "cumple" en todos los registros estará vacío y no se hará nada.

En la segunda función donde usamos un "ensure_one()" trabajamos con un solo registro. En el caso de la primera ejecución cuando se instale el módulo fallará porque en lugar de un solo registro se le están enviando todos en un dataset. Esto provocará una excepción que capturamos. No pasa nada, está previsto así que con un mensaje por la terminal será suficiente.

4. Creando las vistas

En la carpeta views del modelo horoscopo editaremos el archivo views.xml para heredar las vistas ya existentes del módulo de contactos y añadir nuestros nuevos campos.

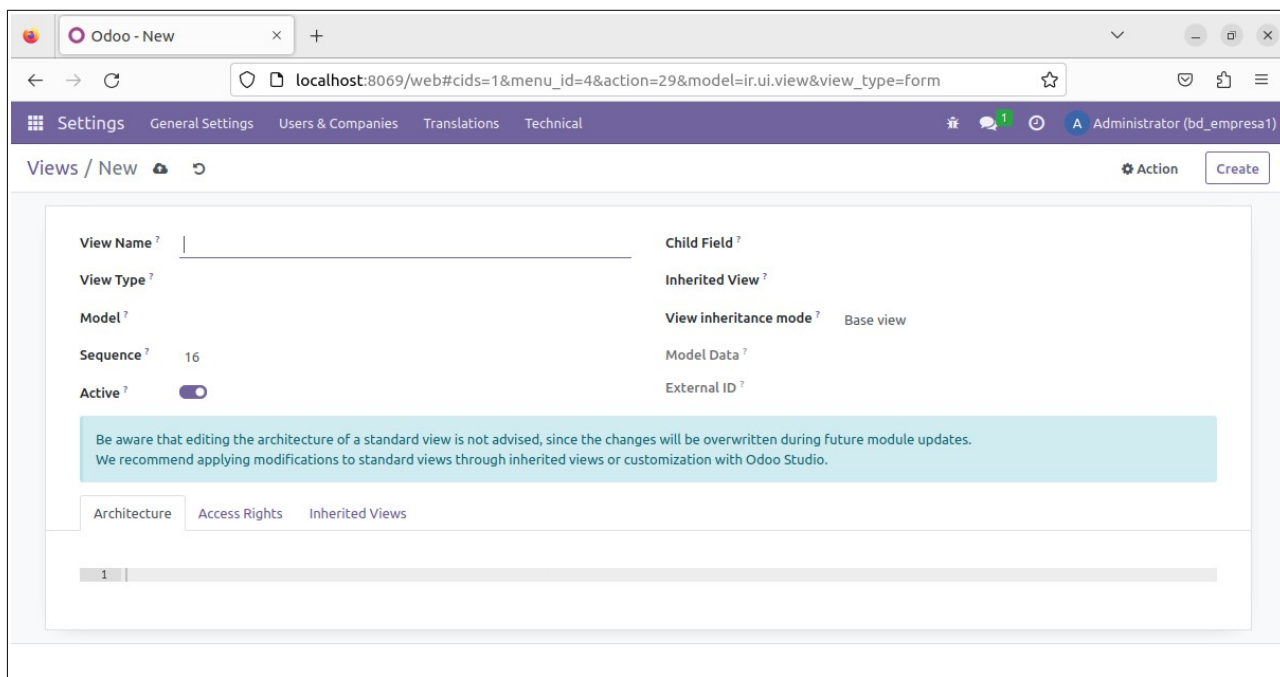
```
<odoo>
<record id="res_partner_horoscopo" model="ir.ui.view">
  <field name="name">res.partner.form.inherit.horoscopo</field>
  <field name="model">res.partner</field>
  <field name="inherit_id" ref="base.view_partner_form" />
  <field name="arch" type="xml">
    <field name="vat" position="after">
      <field name="cumple" />
      <field name="edad" />
      <field name="signo" />
    </field>
  </field>
</record>
</odoo>
```

La línea de la etiqueta record servirá para insertar un nuevo registro con el external id indicado por id en la tabla donde Odoo almacena las vistas (ir.ui.view)

Las 3 siguientes líneas indican el nombre de la vista, el modelo que representa y el external id de la vista de la que se hereda.

Todo lo relacionado con la etiqueta arch tiene que ver con donde posicionar los nuevos campos. Por ahora, en este ejemplo, aparecerán después del campo "vat".

A estas alturas hago un pequeño inciso para indicar que todo esto se puede hacer mediante el entorno gráfico del propio Odoo. Por ejemplo, para crear una nueva vista (o editar alguna existente) debes ir a la aplicación Ajustes, menú Técnico y submenú Vistas.



Odoo - New

localhost:8069/web#ids=1&menu_id=4&action=29&model=ir.ui.view&view_type=form

Settings General Settings Users & Companies Translations Technical Administrator (bd_empresa1)

Views / New

Action Create

View Name ?

View Type ?

Model ?

Sequence ? 16

Active ? ☒

Child Field ?

Inherited View ?

View inheritance mode ? Base view

Model Data ?

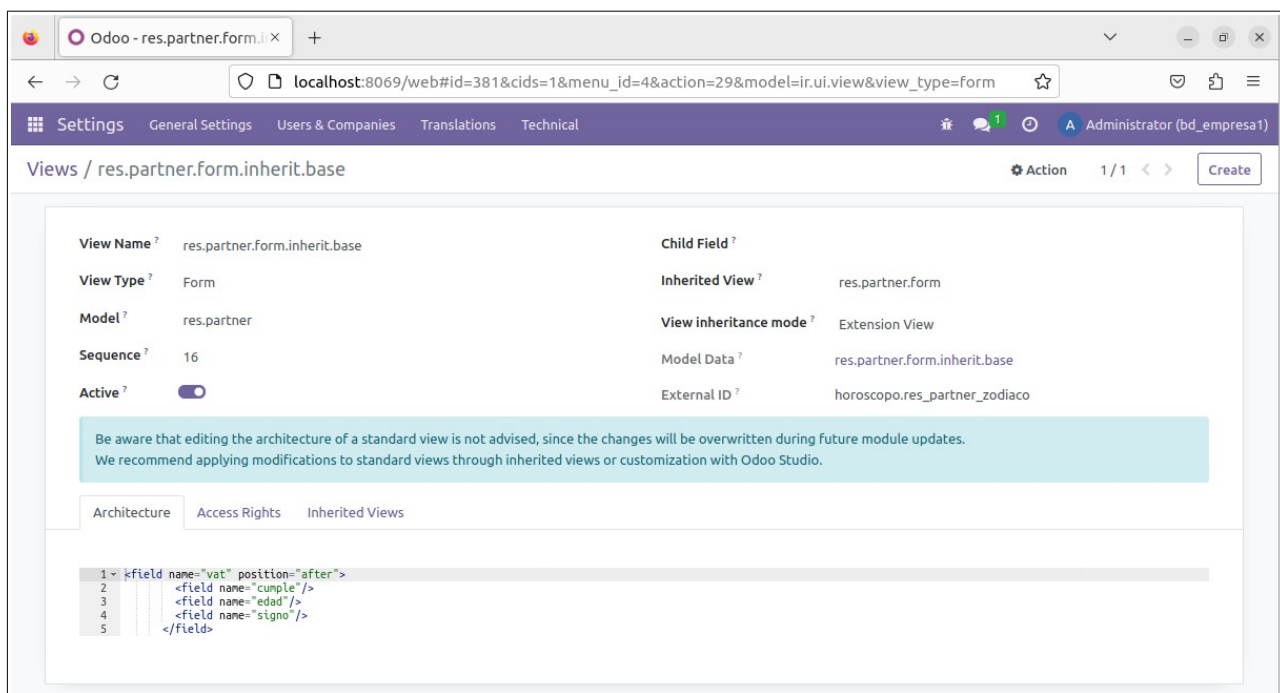
External ID ?

Be aware that editing the architecture of a standard view is not advised, since the changes will be overwritten during future module updates. We recommend applying modifications to standard views through inherited views or customization with Odoo Studio.

Architecture Access Rights Inherited Views

1

A lo largo del curso siempre hemos modificado todo directamente desde el archivo porque pienso que es más didáctico y se tiene el control de donde están todos los elementos. Tanto si creas la vista en el propio archivo como si lo hace a través de la interfaz gráfica el resultado que verás a través del apartado vistas del menú técnico es el siguiente:



Odoo - res.partner.form.inherit.base

localhost:8069/web#id=381&ids=1&menu_id=4&action=29&model=ir.ui.view&view_type=form

Settings General Settings Users & Companies Translations Technical Administrator (bd_empresa1)

Views / res.partner.form.inherit.base

Action 1 / 1 Create

View Name ? res.partner.form.inherit.base

View Type ? Form

Model ? res.partner

Sequence ? 16

Active ? ☒

Child Field ?

Inherited View ? res.partner.form

View inheritance mode ? Extension View

Model Data ? res.partner.form.inherit.base

External ID ? horoscopo.res_partner_zodiaco

Be aware that editing the architecture of a standard view is not advised, since the changes will be overwritten during future module updates. We recommend applying modifications to standard views through inherited views or customization with Odoo Studio.

Architecture Access Rights Inherited Views

```

1 <field name="vat" position="after">
2   <field name="cumple"/>
3   <field name="edad"/>
4   <field name="signo"/>
5 </field>
    
```

Fíjate como los campos mas "administrativos" que definimos al principio del XML de views.xml se muestran en la parte superior. En cambio la parte de "arch" seguimos viéndola como XML en la parte inferior.

5. Primera ejecución

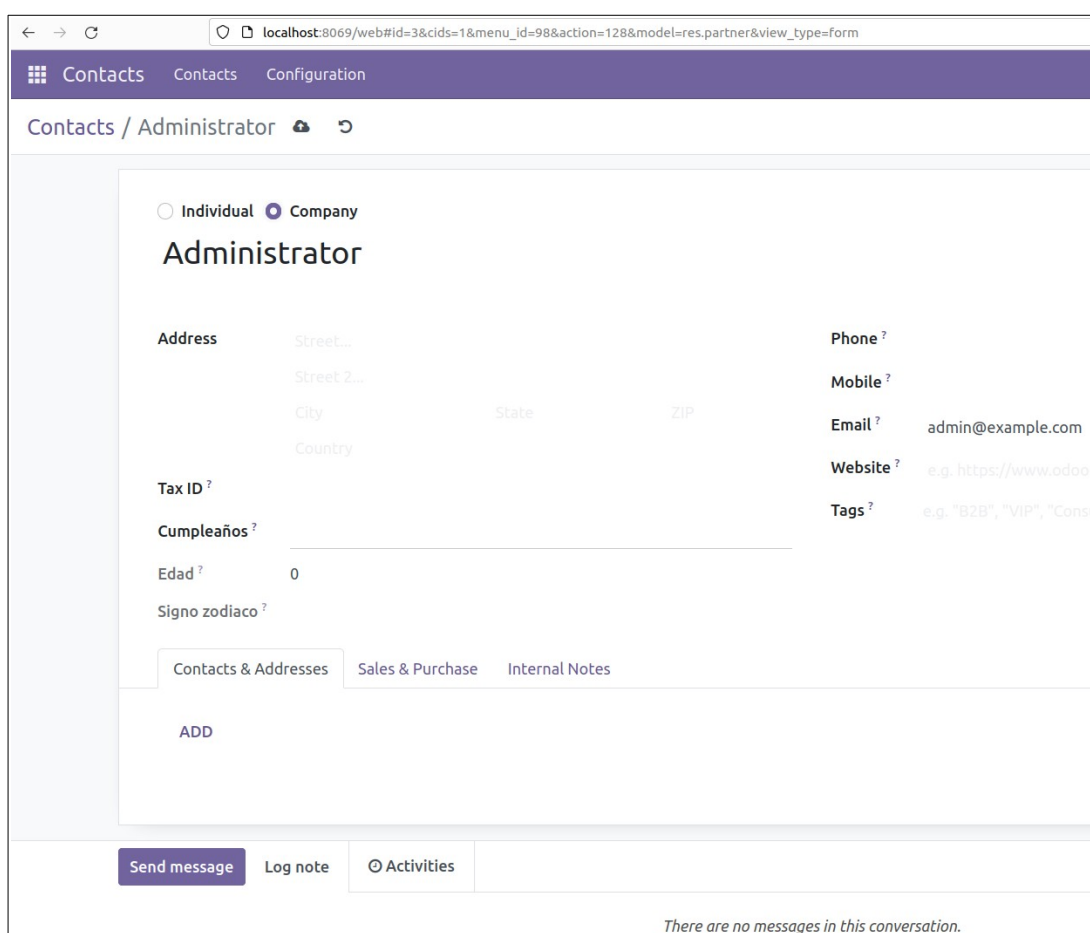
Ahora que ya hemos:

- creado la estructura del módulo dentro de alguna de las carpetas donde Odoo busca módulos
- modificado el archivo `__manifest__.py` para personalizar la información del módulo
- modificado el archivo `models.py` para especificar que heredamos del modelo de contactos (`res.partner`) añadiendo 3 campos, dos de ellos calculados y el código necesario para calcularlo.
- modificado el archivo `views.xml` para heredar las vistas de contactos añadiendo los tres nuevos campos.

Recuerda que aun deberíamos añadir los menús y las acciones para que funcione pero en este caso específico no estamos creando un módulo independiente. Nuestro módulo horóscopo hereda de `res.partner` y allí ya están las acciones y menús creados por lo que no es necesario definirlo de nuevo. Piensa que estamos añadiendo utilidad a algo ya existente.

Si hemos estado haciendo pruebas con el módulo horóscopo es buena idea desinstalarlo. Las columnas añadidas a la tabla de `res.partner` desaparecerán y tendremos un estado de partida limpio.

No está demás reiniciar el servicio para que los cambios en los archivos python se integren en Odoo y también sería interesante refrescar la lista de aplicaciones por si hemos movido la carpeta del módulo. Con esto ya estamos listos para instalarlo por "primera vez"



Una vez instalado, al entrar en el módulo de contactos y seleccionar un registro para poder verlo con vista formulario comprobamos como los tres nuevos campos se han añadido justo después del campo "vat" (Tax ID).

Es el momento de hacer alguna prueba. Inserta nuevos contactos, añade fechas de nacimiento y comprueba que sus campos cambian y que se almacenan. Aprovecha y mira también en la base de datos postgres si los cambios han quedado guardados en la tabla.

6. Refinando las vistas. Formulario

Ahora que nuestro módulo está funcionando parcialmente, vamos a modificar las vistas de `res.partner` para incluir nuestros nuevos campos en el lugar adecuado.

En la vista formulario crearemos una nueva pestaña llamada "Horóscopo" y allí mostraremos los datos. Para colocar los nuevos campos en una vista heredada usaremos XPATH (XML Path) que permite posicionarnos dentro de la estructura jerárquica de un archivo XML donde queramos usando para ello expresiones regulares. En esencia las expresiones se parecen mucho a la ruta que podría tener un fichero dentro de una estructura de carpetas y subcarpetas.

```
<odoo>
<record id="res_partner_horoscopo" model="ir.ui.view">
  <field name="name">res.partner.form.inherit.horoscopo</field>
  <field name="model">res.partner</field>
  <field name="inherit_id" ref="base.view_partner_form" />
  <field name="arch" type="xml">
    <xpath expr="//notebook" position="inside">
      <page string="Zodiaco">
        <group>
          <field name="cumple" string="Fecha de cumpleaños" />
          <field name="edad" string="Edad" />
          <field name="signo" string="Signo del zodiaco" />
        </group>
      </page>
    </xpath>
  </field>
</record>
</odoo>
```

Si analizar el XML verás que en la expresión del xpath pedimos que busque la etiqueta `notebook` y que se posicione dentro de ella para pegar las siguientes líneas de una nueva página.

Cuando instalemos el módulo la vista formulario debería ser como la siguiente:

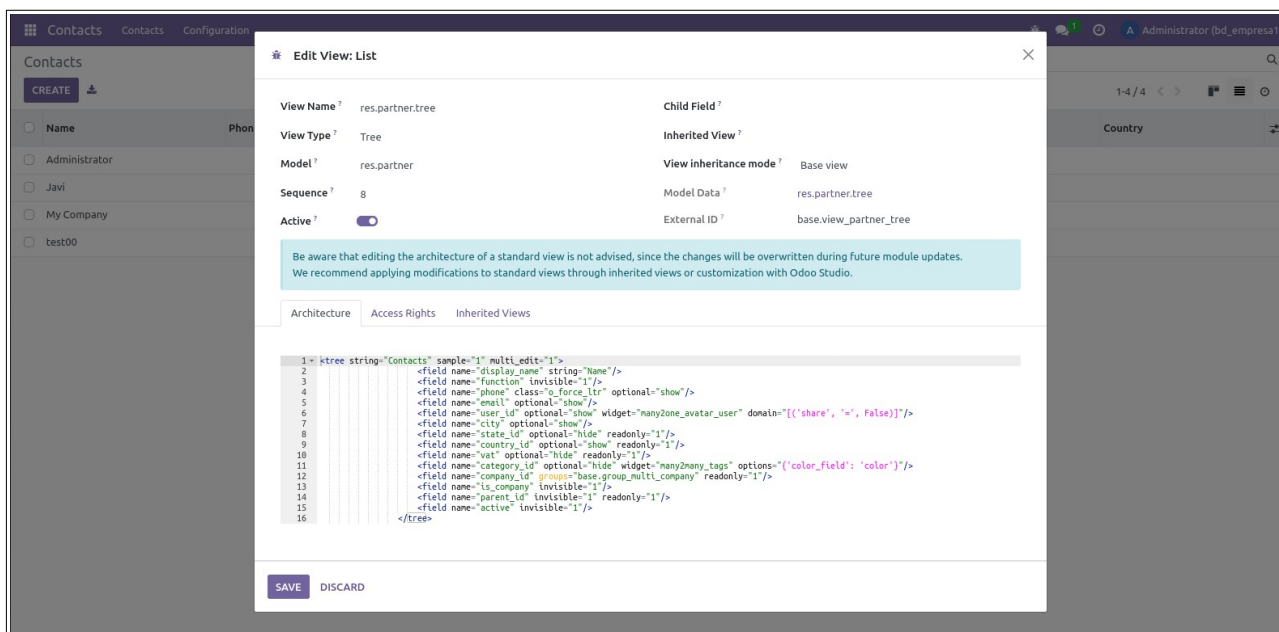
The screenshot shows the 'Contacts / Administrator' form in Odoo. At the top, there are tabs for 'Contacts', 'Configuration', and 'Action'. Below the tabs, there are radio buttons for 'Individual' (selected) and 'Company'. The form title is 'Administrator'. There is a 'Company Name...' field. Below that, there are fields for 'Contact' (Street, Street 2, City, State, ZIP, Country) and 'Tax ID'. To the right, there are fields for 'Job Position', 'Phone', 'Mobile', 'Email', 'Website', 'Title', and 'Tags'. At the bottom, there are tabs for 'Contacts & Addresses', 'Sales & Purchase', 'Internal Notes', and 'Zodiaco'. The 'Zodiaco' tab is active, showing fields for 'Fecha de cumpleaños', 'Edad', and 'Signo del zodiaco'.

7. Refinando las vistas. Lista

Dentro de la aplicación de contactos podemos ver los distintos registros en forma de lista. Puedes activar esta vista pinchando en el icono de la esquina superior derecha que tiene forma de 4 líneas horizontales.

Esta vista lista muestra información parcial de varios registros y nosotros vamos a querer editarla para incluir uno de nuestros campos en ella. Incluiremos el campo "signo de zodiaco" justo después de nombre, en la segunda columna.

En primer lugar debemos averiguar como se llama esta vista para poder heredar de ella y modificarla. Esto lo hacemos pinchando en el icono del desarrollador (el que tiene forma de bicho) y seleccionando "edit view: list".



Edit View: List

View Name: res.partner.tree
 View Type: Tree
 Model: res.partner
 Sequence: 8
 Active: ☒
 Child Field:
 Inherited View:
 View inheritance mode: Base view
 Model Data: res.partner.tree
 External ID: base.view_partner_tree

Be aware that editing the architecture of a standard view is not advised, since the changes will be overwritten during future module updates. We recommend applying modifications to standard views through inherited views or customization with Odoo Studio.

Architecture Access Rights Inherited Views

```

1 <tree string="Contacts" sample="1" multi_edit="1">
2   <field name="display_name" string="Name"/>
3   <field name="function" invisible="1"/>
4   <field name="phone" class="o_force_ltr" optional="show"/>
5   <field name="email" optional="show"/>
6   <field name="user_id" optional="show" widget="many2one_avatar_user" domain="[[('share', '=', False)]]"/>
7   <field name="city" optional="show"/>
8   <field name="state_id" optional="hide" readonly="1"/>
9   <field name="country_id" optional="show" readonly="1"/>
10  <field name="vat" optional="hide" readonly="1"/>
11  <field name="category_id" optional="hide" widget="many2many_tags" options="{ 'color_field': 'color' }"/>
12  <field name="company_id" groups="base.group_multi_company" readonly="1"/>
13  <field name="is_company" invisible="1"/>
14  <field name="parent_id" invisible="1" readonly="1"/>
15  <field name="active" invisible="1"/>
16 </tree>
    
```

SAVE DISCARD

Aquí ya podemos ver que el external id de esta vista es `base.view_partner_tree`, será la vista de la que vamos a heredar. Ya lo sabemos pero aquí también se puede apreciar que el modelo que se muestra en esta vista es el de `res.partner`. También vemos/confirmamos que se trata de una vista tipo "tree" que es la propia de las listas.

Por último, en la parte de abajo vemos el XML en el que se define la lista. Es muy sencillo de interpretar y fácil de localizar donde queremos insertar nuestro nuevo campo.

Para modificar esta vista podemos crear un otro nuevo archivo XML en la carpeta views similar al que usamos al editar el formulario en el apartado anterior. En este caso recuerda que debes incluirlo en el archivo `__manifest__.py` de la raíz del módulo para que sea incluido.

Yo voy a incluirlo en el mismo archivo que la modificación del formulario, me quedará así:

```

<odoo>
<record id="res_partner_zodiaco_form" model="ir.ui.view">
  <field name="name">res.partner.form.inherit.base</field>
  <field name="model">res.partner</field>
  <field name="inherit_id" ref="base.view_partner_form" />
  <field name="arch" type="xml">
    <xpath expr="//notebook" position="inside">
      <page string="Zodiaco">
        <group>
          <field name="cumple" string="Fecha de cumpleaños"/>
          <field name="edad" string="Edad"/>
          <field name="signo" string="Signo del zodiaco"/>
        </group>
      </page>
    </xpath>
  </field>
</record>

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



```
<field name="name">res.partner.tree.inherit.base</field>
<field name="model">res.partner</field>
<field name="inherit_id" ref="base.view_partner_tree" />
<field name="arch" type="xml">
    <xpath expr="//tree/field[@name='display_name']" position="after">
        <field name="signo" string="Signo del zodiaco"/>
    </xpath>
</field>
</record>

</odoo>
```

Analizando este código XML puedes encontrar en la primera etiqueta record la modificación del formulario y en la segunda etiqueta record la modificación de la lista. Fíjate que he cambiado el nombre del id de cada registro que se va a insertar en la tabla de las vistas (ir.ui.view) para que no se pisen y sean legibles.

En ambas se ha utilizado xpath para posicionarnos dentro del XML heredado y después se ha insertado el nuevo código.

8. Refinando las vistas. Kanban

Por último vamos a modificar la vista kanban en la que se ven los registros en forma de tarjeta con información muy reducida.

La vista kanban es un poco más complicada pero seleccionando el "edit view:kanban" del menú desarrollador se puede entender donde queremos colocar el campo. Para ello usamos XPATH e vamos definiendo la ruta.

Nuevamente podemos elegir si queremos definir esta vista en un nuevo archivo XML que tendremos que añadir al __manifest__.py de nuestro módulo o bien aprovechar el views.xml que ya venimos usando. Yo he optado por continuar el archivo y dejar las 3 vistas definidas en el mismo fichero.

El código completo de las 3 vistas quedaría así:

```
<odoo>
<record id="res_partner_zodiaco_form" model="ir.ui.view">
    <field name="name">res.partner.form.inherit.base</field>
    <field name="model">res.partner</field>
    <field name="inherit_id" ref="base.view_partner_form" />
    <field name="arch" type="xml">
        <xpath expr="//notebook" position="inside">
            <page string="Zodiaco">
                <group>
                    <field name="cumple" string="Fecha de cumpleaños"/>
                    <field name="edad" string="Edad"/>
                    <field name="signo" string="Signo del zodiaco"/>
                </group>
            </page>
        </xpath>
    </field>
</record>
```

```

<record id="res_partner_zodiaco_tree" model="ir.ui.view">
  <field name="name">res.partner.tree.inherit.base</field>
  <field name="model">res.partner</field>
  <field name="inherit_id" ref="base.view_partner_tree" />
  <field name="arch" type="xml">
    <xpath expr="//tree/field[@name='display_name']" position="after">
      <field name="signo" string="Signo del zodiaco"/>
    </xpath>
  </field>
</record>

<record id="res_partner_zodiaco_kanban" model="ir.ui.view">
  <field name="name">res.partner.kanban.inherit.base</field>
  <field name="model">res.partner</field>
  <field name="inherit_id" ref="base.res_partner_kanban_view" />
  <field name="arch" type="xml">
    <xpath expr="//kanban/templates/t/div/div[2]/div/strong"
position="after">
      <p></p>
      <field name="signo"/>
    </xpath>
  </field>
</record>

</odoo>
    
```

9. El reto.

Llega el momento de incorporar el código que calcule la edad y el código que calcule el signo del zodiaco en el archivo models.py. Suerte!