Comunitea

# Formación Técnica

# Odoo

Comunitea

# Instalación desde código y creación de la primera base de datos.

# Requisitos (I)

1- Git

- sudo apt-get install git

2- Postgresql

- sudo apt-get install postgresql, pgadmin3

3- Buildout

- Añadir en /etc/apt/sources.list la línea:
  deb http://apt.anybox.fr/openerp common main
- Update: sudo apt-get update
- sudo apt-get install openerp-server-system-build-deps
- sudo apt-get install python-setuptools

Para firmar el repositorio:

wget http://apt.anybox.fr/openerp/pool/main/a/anybox-keyring/anybox-keyring_0.2_all.deb

Comunitea

# Dependencias (II)

4- Wkhtmltopdf

- wget

http://download.gna.org/wkhtmltopdf/0.12/0.12.1/wkhtmltox-0.12.1_linux-trusty-amd64.deb

- sudo dpkg -i wkhtmltox-0.12.1_linux-trusty-amd64.deb

Comunitea

**Postgresql configuración:**

1- Creación de usuario (peer login, por defecto):

```
sudo su - postgres -c "createuser -s $USER"
```

2- Creación de usuario (md5 login, cambiando fichero pg_hba.conf en /etc/postgresql/10/main/pg_hba.conf):

```
sudo su – postgres
createuser --createdb --username postgres --no-createrole --pwprompt USER
```

Comunitea

**Configuración buildout para desarrollo:**

1- Descargar el repositorio

- git clone -b 10.0 https://github.com/Comunitea/odoo_buildout_base.git

2- Configurar ficheros para desarrollo

- Copiar buildout.cfg a devel_buildout.cfg
- Copiar odoo.cfg a devel_odoo.cfg
- devel_odoo.cfg: odoo_unaccent = False, odoo_pg_path = /usr/bin/, postgres_db_name = base_de_datos, postgres_port=5432
- devel_buildout.cfg: en extends: odoo.cfg → devel_odoo.cfg, en parts: borramos todo lo que hay después de odoo

3- Ejecutar:

- sudo easy_install virtualenv
- virtualenv sandbox --no-setuptools
- sandbox/bin/python bootstrap.py
- bin/buildout -c devel_buildout.cfg

4- Arrancar

- bin/start_odoo -d base_de_datos (Para actualizar: --update module)

Comunitea

# Creación y configuración cuenta Github

- Crear cuenta o loguearse en Github
- Crear o copiar clave pública y asociarla a Github
    - Comprobar si existe: ls -al ~/.ssh
    - Sino existe:
        - ssh-keygen -t rsa -b 4096 -C "your_email@example.com"
        - eval "$(ssh-agent -s)"
        - ssh-add ~/.ssh/id_rsa
    - Añadir la clave a Github

Comunitea

# Instalación IDE programación:

Hay varias IDEs de programación disponibles para trabajar con Python, por ejemplo PyCharm (code compleation, debugger, heavy), Eclipe Pydev (code completion, debugger, Eclipse plugins, heavy, Java), Sublime Text (customizable, light) y Geany (customizable, helpful, light) entre otros. Se puede consultar una gran comparativa en: http://stackoverflow.com/questions/81584/what-ide-to-use-for-python

Comunitea

# Configuración e instalación Geany:

sudo apt-get install geany geany-plugins
- En Herramientas/Administrador de complementos seleccionamos, el Añadido y Visor de árbol en las preferencias los configuramos a nuestro gusto.
- En Editar/Preferencias, configuramos la ruta de inicio, la columna de salto de línea a 79 caracteres, borrar espacios extra al crear una nueva línea, sangría con tabuladores y espacios con 4 caracteres de longitud, borrar espacios y tabulaciones al final y reemplazar tabulaciones por espacios.
- Cargar script de comprobación de código en la tecla F9 con seguimiento de errores.

Comunitea

# Programando un módulo

# Estructura de un módulo:

- Se compone siempre de un fichero **__manifest__.py** donde se define la información base del módulo y de un fichero **__init__.py** donde se importan todos los ficheros python del módulo.
- A veces también se incluye un fichero **README.md** con documentación del módulo.
- Dentro de una carpeta **models**, se define un fichero por cada modelo que se cree o extienda.
- Dentro de una carpeta **wizard** se definen los asistentes y sus vistas.
- Dentro de una carpeta **views** se definen las vistas de los modelos.
- Dentro de una carpeta **data**, se añaden xml con datos a importar que pueden ser desde maestros a datos de configuración.
- Dentro de una carpeta **security** se crean los ficheros que definen los accesos a los modelos.
- Dentro de la carpeta **controllers** se definen las distintas peticiones que se programen contra el framework web.
- Dentro de la carpeta **static**, nos podemos encontrar con ficheros estáticos dentro de subcarpetas **js**, **img**, **css**, normalmente son ficheros que se usan desde el framework web.

Comunitea

# __manifest__.py:

El formato es idéntico en todos los módulos de Odoo.
Es el fichero que usa Odoo para reconocer que es un módulo y que se puede instalar.
Se compone de:

- name: Nombre del módulo
- summary: Descripción corta del módulo
- description: Descripción larga de la funcionalidad del módulo, el README.md la substituye.
- author: Empresa autora
- website: Web del autor o de referencia.
- category: Tipo de módulo.
- version: Número de versión.
- depends: Módulos de los que depende
- data: Ficheros xml y csv que se desean importar como datos.
- demo: ficheros xml y csv que se desean importar como demostración.
- license: Identificador de licencia del módulo, por defecto AGPL-3

Comunitea

# ORM – Modelos (Tablas):

Odoo viene con una capa ORM por defecto que nos sirve para interacturar con la base de datos sin usar sql.
Los modelos se declaran como clases de Python heredando la clase **Model** del fichero **models** de Odoo

```
from odoo import models
Class MyModel(models.Model):
    _name = “my.model”
```

El modelo se identifica por un atributo de clase denominado **_name**, en el caso de que sea una extensión de un modelo ya existente este atributo se cambia por **_inherit**. El _name se usa para crear la tabla en la base de datos substituyendo los . por _.

Comunitea

# ORM – Campos (Columnas) (I):

- Los campos se definen como atributos de clase usando una llamada al fichero fields de Odoo y a la clase que corresponda según el dato que estemos representando.
- Las clases disponibles son Char, Text, Selection, Integer, Float, Monetary, Many2one, One2many, Many2many, Date, Datetime, Binary, Html, Reference y Boolean.
- Estos campos se construyen con unos atributos comunes, entre ellos string, required, readonly y help.
- Luego cada campo tiene sus atributos específicos.
- Todo modelo tiene que tener un campo name, sino se deberá definir un atributo de clase _rec_name apuntando hacia un campo de los del modelo que hará la función de nombre.

Comunitea

# Ejercicio 1

Ejercicio 1
Crear un módulo de nombre openacademy con la estructura básica __manifest__.py e __init__.py

# Ejercicio 2

Definamos un modelo “Course” con título y descripción, el título debe ser obligatorio.
Instalar el módulo

Comunitea

# ORM – Carga de datos:

Las vistas, informes, datos maestros, configuraciones etc... se cargan con xmls normalmente, cada registro cargado por xml se convierte en un registro de un modelo (tabla) de Odoo, por lo tanto, la definición de estos registros tiene un formato común.

```
<odoo>
    <data>
        <record model="{model name}" id="{record identifier}">
            <field name="{a field name}">{a value}</field>
        </record>
    </data>
<odoo>
```

model: Se corresponde con el _name del modelo al que el estamos cargando el registro.
id: Es una cadena única dentro del módulo que identifica al registro en la base de datos, para actualizaciones futuras.
field_name: Es el nombre del campo que se desea escribir.

Comunitea

# Ejercicio 3

Definamos un modelo “CourseType” con código y descripción, ambos obligatorios.
Carguemos un par de tipos desde un xml de datos.
Actualizar el módulo

Comunitea

# ORM – Acciones y menús:

Se definen también en xml igual que antes excepto el menú que aunque se podrían definir igual que cualquier otro registro tiene un formato de definición más sencillo.

```
<record model="ir.actions.act_window" id="{record.identifier}">
    <field name="name">Nombre Acción</field>
    <field name="res_model">{modelo}</field>
    <field name="view_mode">tree,form</field>
</record>
<menuitem id="{record.identifier}" parent="{parent_record.identifier}"
name="Nombre Menú" sequence="10" action="{action.identifier}"/>
```

Comunitea

# Ejercicio 4

Crear un menú general Open Academy, debajo de nuevo un menú Courses y otro Configuration, en Configuration se crearía uno de nombre Courses types que abriría el modelo de tipos de cursos y debajo de Courses se crearía otro Courses que abriría el modelo de cursos.

Comunitea

# ORM – Vistas:

Se definen también en xml como hasta ahora, se corresponden con el modelo ir_ui_view.

```
<record model="ir.ui.view" id="view_id">
    <field name="name">view.name</field>
    <field name="model">object_name</field>
    <field name="priority" eval="16"/>
    <field name="arch" type="xml">
        <!-- view content: <form>, <tree>, <graph>, ... -->
    </field>
</record>
```

Comunitea

# ORM – Vistas Lista:

Muestran registros en una tabla.

Se define dentro de <tree></tree> listando todos los campos que se desean mostrar.

```
<tree string="Idea list">
    <field name="name"/>
    <field name="inventor_id"/>
</tree>
```

# ORM – Vistas Formulario (1):

- Se usan para crear, ver en detalle o editar registros concretos.
- Se define dentro de <form></form> listando todos los campos que se desean mostrar dentro de ciertas estructuras de cara a la presentación como pueden ser group, notebook, page, div, sheet …

```
<form string="Idea form">
    <group colspan="4">
        <group colspan="2" col="2">
            <separator string="General stuff" colspan="2"/>
            <field name="name"/>
            <field name="inventor_id"/>
        </group>
        <group colspan="2" col="2">
            <separator string="Dates" colspan="2"/>
            <field name="active"/>
            <field name="invent_date" readonly="1"/>
        </group>
        <notebook colspan="4">
            <page string="Description">
                <field name="description" nolabel="1"/>
            </page>
        </notebook>
        <field name="state"/>
    </group>
</form>
```

Comunitea

# Ejercicio 5

Crear la vista formulario y lista del modelo course, mostrando los campos título y descripción.

Comunitea

# ORM – Vistas Formulario (2):

- Los formularios de Odoo pueden usar etiquetas html y llamar a clases de css.

```
<form string="Idea Form">
    <header>
        <button string="Confirm" type="object" name="action_confirm"
                states="draft" class="oe_highlight" />
        <button string="Mark as done" type="object" name="action_done"
                states="confirmed" class="oe_highlight"/>
        <button string="Reset to draft" type="object" name="action_draft"
                states="confirmed,done" />
        <field name="state" widget="statusbar"/>
    </header>
    <sheet>
        <div class="oe_title">
            <label for="name" class="oe_edit_only" string="Idea Name" />
            <h1><field name="name" /></h1>
        </div>
        <separator string="General" colspan="2" />
        <group colspan="2" col="2">
            <field name="description" placeholder="Idea description..." />
        </group>
    </sheet>
</form>
```

Comunitea

# ORM – Vistas Búsqueda:

Permite añadir filtros y agrupadores sobre la vista lista.

Se define dentro de <search></search> listando todos los campos que se desean mostrar.

```
<search>
    <field name="name"/>
    <field name="inventor_id"/>
    <group name="group_by" string="Group By">
        <filter string="Inventor" context="{'group_by': 'inventor_id'}"/>
   </group>
</search>
```

Comunitea

# Ejercicio 6

Crear una vista de búsqueda sobre el modelo de cursos, por cualquiera de sus campos.

# ORM – CRUD:

**Insert**: Se corresponde con el método **create** de Odoo, recibe como argumentos self y vals que es un diccionario con todos los valores a insertar en la tabla y devuelve el objeto que acaba de crear.

**Select**: Se pueden utilizar varios métodos si queremos hacer un where se utilizará **search** y se le pasaría como argumentos un domain nos devolverá todos los objetos que encuentre que cumplan el domain. Si queremos convertir a objeto un id de un registro utilizamos **browse** pasándole como argumento el id o ids que queremos instanciar, nos devolverá los objetos instanciados, si queremos obtener el valor de alguno de los campos se utiliza **read** se la pasa argumento el id o ids de los que queremos obtener información y un listado con los campos que queremos que nos devuelva.

**Update**: Se corresponde con el método **write** de Odoo, recibe como argumento self y vals, en este caso self será un recordset, se puede también actualizar un objeto con una asignación simple: self.name = “Nuevo nombre”. Devuelve True.

**Delete**: Se corresponde con el método **unlink** de Odoo, que recibe como argumento sólo self, pero este self será un recorset. Devuelve True

Comunitea

# Relaciones entre modelos:

Se puede relacionar un modelo con otro con 3 tipos de campos Many2one, One2many y Many2many.

- Many2one(other_model, string, ondelete='set null') Se corresponden con Fks

    *print foo.other_id.name*

- One2many(other_model, fk_name, string) El lado contrario a una FK

    *for other in foo.other_ids:*

        *print other.name*

- Many2many(other_model, rel_table_name, model_fk_name, other_model_fk_name, string) Tabla intermedia entre dos modelos vista como un one2many desde el modelo que la define.

    for other in foo.other_ids:

        print other.name

Comunitea

# Ejercicio 7

Crear un modelo Session con sus vistas y menús, representa la impartición de un curso.
Una sesión tiene un nombre, fecha, duración, curso, profesor (relación con usuarios) y número de plazas todos obligatorios.

Añadir también un campo responsable (relación con usuarios) y sesiones (one2many) al curso, estos campos también se debe añadir a las vistas del curso.

Añadir un campo Asistentes también en Session (many2many contra clientes). Añadir este campo a la vista de sesiones.

Comunitea

# Herencia de Modelos:

Odoo soporta 2 tipos de herencia

**Herencia tradicional**: Utilizando _inherit, nos permite añadir campos a un modelo, añadirle/cambiarle atributos a campos ya existentes, añadir constraints, añadir/sobrescribir/extender métodos y extender modelos con funcionalidades dadas por los modelos heredados.
**Herencia por delegación**: Utilizando _inherits. Nos permite asociar un modelo a través de una FK con otro, usando desde el modelo hijo campos del padre transparentemente, si no se rellena explícitamente la FK de relación se creará un modelo padre de forma automática para cada hijo.

Comunitea

# Herencia de  Vistas:

Las vistas de Odoo se pueden sobreescribir, referenciando al id original o se pueden extender a través de la referencia hecha en el campo inherit_id

Si se quiere extender una vista hay que marcarle el punto de inicio desde el que se va a hacer la modificación para esto podemos utilizar rutas xpath o llamadas a elementos ya definidos en la vista original añadiendo un atributo position (after, before, inside, attributes, replace)

```
<record id="idea_category_list2" model="ir.ui.view">
    <field name="name">id.category.list2</field>
    <field name="model">ir.ui.view</field>
    <field name="inherit_id" ref="id_category_list"/>
    <field name="arch" type="xml">
        <!-- find field description inside tree, and add the field
             idea_ids after it -->
        <field name="name' position="after">
          <field name="idea_ids" string="Number of ideas"/>
        </field>
    </field>
</record>
```

Comunitea

# Ejercicio 8

Heredar el modelo de res_partner y añadir una campo booleano de nombre profesor, añadirlo en la vista de empresas debajo de los campos de cliente y proveedor.

Añadir este mismo campo a la vista formulario de usuarios.

Comunitea

# Dominios:

Es la estructura que usa Odoo para filtrar sobre un modelo, a modo de un where de Sql. Es una lista de tuplas de 3 valores cada una. (nombre de campo, operador,  valor)

```
[('product_type', '=', 'service'), ('unit_price', '>', 1000)]
```

Por defecto entre tupla y tupla se considera un AND si quisiéramos indicar un OR, tendríamos que anteponer a un par  de tuplas '|', también se puede indicar el AND explícitamente con '&' y el NOT con '!' de la misma forma.

```
['|',('product_type', '=', 'service'),'!', '&',
  ('unit_price', '>=', 1000),('unit_price', '<', 2000)]
```

Comunitea

# Ejercicio 9

En las Sesiones sólo debe permitir escoger profesores con la marca de profesor puesta.
En el listado de asistentes de una sesión sólo debe dejar coger empresas con la marca de cliente puesta.

Comunitea

# Self:

self aparte de ser la palabra reservada en Python para acceder a la clase dentro de la misma, representa una colección ordenada de objetos en Odoo, un recordset, soporta las operaciones estándar de Python, como len(self), iter(self), recs1 + recs2. Cada objeto dentro de self se puede inspeccionar usando la notación de puntos Ej: record.name

Comunitea

# Api (1):

Es una clase de Odoo que nos permite usar decoradores en nuestros métodos para que modifiquen su comportamiento.

La firma por defecto de los métodos de OpenERP se compone de varios argumentos que se usaban continuamente:

cr: Cursor a la base de datos, mantiene el objeto conexión de la base de datos, gestiona las transacciones contra base de datos y nos permite si queremos hacer sqls directos usando su método execute().

uid: Id del usuario activo

Ids: Lista de ids del modelo activo sobre los que se está ejecutando el método.

Context: Diccionario con ciertos valores de sesión (idioma por ejemplo) o valores dinámicos que pudiera interesar pasar desde una vista.

Comunitea

# Api (2):

En la versión 8.0 esto se simplificó y ahora la mayoría de métodos sólo reciben el argumento self, que vimos anteriormente, desde este se puede seguir accediendo a los argumentos anteriores pero pasaron a un segundo plano.

- self._cr o self.env.cr
- self._uid o self.env.uid o self.env.user.id
- self._context o self.env.context
- self.ids

Comunitea

# Api (3):

@api.multi
El argumento self siempre va a ser una lista de objetos aunque sólo contenga uno, son todos los objetos sobre los que se ejecuta el método.

@api.one
Itera de forma automática en self y llama tantas veces como objetos hubiera en el self original. Marcado como “deprecated” en Odoo 9.0

@api.model
Tiene la misma funcionalidad que api.multi pero se usa en métodos que no reciben la típica firma de cr, uid, ids, context.

Comunitea

# Api (4):

@api.depends('campo')
Puede ir acompañado de api.one o api.multi se usa normalmente en campos función para que el cambio en alguno de los campos de los que depende dispare el recálculo del campo función.

Comunitea

# Ejercicio 10

Añadir al modelo de Sesión un campo calculado, que nos informe del % de ocupación y mostrarlo tanto en la vista lista como formulario sólo lectura.

** Los campos función se definen igual que los demás pero indicando un atributo más de nombre compute que tendrá como valor el nombre de la función de cálculo.

Comunitea

# self.env

Uno de los atributos más utilizados de self es self.env desde este objeto podemos acceder al usuario activo, cursor de conexión a la base de datos, contexto, pool de modelos e id interno de referencias xml.

self.env[model_name] Nos devuelve una acceso del modelo pedido
self.env.ref(xml_id) No devuelve el objeto referenciado con ese id

Comunitea

# Valores por defecto:

Se pueden definir valores por defecto en los campos de los modelos con un atributo default=”valor” siempre que el valor sea un valor estático, si fuera calculado hay que utilizar los generadores de funciones dinámicas de Python (lambda).

```
name = fields.Char(default="Unknown")
user_id = fields.Many2one('res.users', default=lambda self: self.env.user)
```

Comunitea

# Ejercicio 11

Añadir un campo booleano nuevo a la sesión de nombre active por defecto a True y poner un valor por defecto al campo fecha, con la fecha actual.

** Odoo proporciona dos funciones para obtener la fecha actual sin tener que tirar directamente de ningún paquete de python como podría ser datetime o time, estas funciones son:
fields.Date.context_today - Nos devuelve una fecha como cadena en el formato %Y-%m-%d que es el esperado por el ORM
fields.Datetime.now - Nos devuelve una fecha y hora como cadena en el formato “%Y-%m-%d %H:%M:%S” que es el esperado por el ORM

** active es un campo especial ya que si está desmarcado hace que el registro se oculte por defecto de la vista lista de Odoo. Sólo se podrán ver los registros desactivados si se buscan explícitamente

Comunitea

# Onchange:

Método de clase que se comporta como un evento de cambio que cada vez que cambia el valor del campo que vigila se dispara y puede hacer cambios dinámicos en un formulario en edición o creación.

Se define con el decorador @api.onchange(campos que vigila) puede cambiar valores de self que será el modelo que estamos creando o editando o puede lanzar un aviso devolviendo un warning con la forma:

```
    return {
     'warning': {
         'title': "Something bad happened",
         'message': "It was very bad indeed",
     }
}
```

También puede cambiar un domain de vista de forma dinámica.

Comunitea

# Ejercicio 12

Añadir un onchange en los campos número de asientos y asistentes de la sesión para que salte un aviso si el número de asientos es negativo o se pusieran más asistentes que asientos.

Comunitea

# Constraints (1):

Son eventos que se lanzan en cada escritura de un modelo a base de datos y pueden ser comprobaciones de la información introducida como que no esté duplicada o que esté en cierto rango de valores, no deja guardar el registro hasta que se solucione.

Los constraints se pueden definir de dos formas, a través de Postgresql o a través de Python.

En python se definen en un método con el decorador @api.constrains(campo a vigilar) si queremos que falle lanzaremos una excepción de Odoo con raise.

** Las excepciones de Odoo se crean usando el fichero exceptions de odoo y llamando a la clase que nos interese UserError, ValidationError, MissingError, AccessError  en el caso de constraints:

```
raise exceptions.ValidationError(_("Mensaje"))
```

Comunitea

# Constraints (2):

Las constraints de Postgresql se definen con el atributo de clase _sql_constraints se define como una lista de tupas de 3 valores (nombre, definicion sql, mensaje)

```
 _sql_constraints = [
        ('name_unique',
         'UNIQUE(name)',
         "The course title must be unique"),
    ]
```

Comunitea

# Ejercicio 13

Añadir un constraint de python que compruebe que el profesor no está entre los asistentes a una sesión y otro para que el valor de número de asientos no pueda nunca ser negativo.

Añadir dos constraints de sql para que el nombre del curso sea único y no sea igual que lo introducido en descripción.

Comunitea

# Duplicar registros:

Odoo por defecto nos permite duplicar cualquier registro desde su vista formulario, pero hay campo que nos puede interesar no duplicar nunca, en ese caso al definirlos le añadimos un atributo copy=False y otros que vamos a querer que los duplique pero con alguna peculiaridad, para eso tendremos que extender el método copy.

Comunitea

# Ejercicio 14

Al poner único el valor del campo nombre del curso acabamos de impedir que se pueda duplicar un curso, por lo que necesitamos extender el método copy y duplicar el campo de nombre pero añadiendo al nombre copy.

No copiar nunca asistentes, ni fecha de sesión.

# Ejercicio 15

Mejoras funcionales

- Añadir un campo estado a las Sesiones, En Preparación, Inscripción abierta, Terminada y Cancelada. Por defecto En Preparación.
- Crear botones para cambiar de estados.
- Sólo se permiten borrar sesiones en preparación o canceladas.
- No se puede cancelar en estado finalizado.
- Añadir un campo producto al curso, sólo servicios.

Comunitea

# Informes Qweb (I)

Como los menuitem tiene una definición simplificada:

```
<report
      id="report_id"
      model="odoo.model"
      string="Name"
     report_type="qweb-pdf"
     name="module.report_name"
     file="module.report_name"/>
```

A parte de los atributos que se muestran en el ejemplo, existen otros:

- report_name = Nombre de salida del pdf
- attachment_use = Si se va a adjuntar al imprimirse
- attachment = Si de adjunta nombre del adjunto
- paperformat = Si se quiere usar un tamaño de papel distinto al por defecto A4

Comunitea

# Informes Qweb (II)

Un informe qweb se define mínimamente con:

```
<template id="report_name">
    <t t-call="report.html_container">
        <t t-foreach="docs" t-as="o">
            <t t-call="report.external_layout">
                <div class="page">
                    <h2>Report title</h2>
                    <p>This object's name is <span t-field="o.name"/></p>
                </div>
            </t>
        </t>
    </t>
</template>
```

docs: Son los registros seleccionados al imprimir

Comunitea

# Informes Qweb (III)

Internamente es código html y css por lo general, con algunas cosas que añade el motor de plantillas qweb:

t-call: Permite llamar a otra plantilla web para extender el documento actual.
t-foreach: Permite recorrerse una lista de objetos
t-if: Permite hacer condicionales.
t-esc: Evalúa directamente código python
t-field: Permite llamar a un campo del registro

Odoo tiene muchos estilos cargados por defecto que se pueden utilizar en los informes, se pueden ver en static/src/css de los módulos base, web y report.

# Ejercicio 16

Hacer un informe sobre sesiones que muestre información del curso y sesión y una tabla con todos los participantes.

# Traducciones (I)

Para que los informes qweb sean traducibles hay que llamarlos desde una vista padre con la siguiente estructura:

```
<template id="report_name">
    <t t-call="report.html_container">
        <t t-foreach="docs" t-as="doc">
                <t t-call="module.report_name_document" t-lang="doc.partner_id.lang"/>
        </t>
    </t>
</template>

<template id="report_saleorder_document">
    <t t-set="doc" t-value="doc.with_context({'lang':doc.partner_id.lang})" />
    <t t-call="report.external_layout">
        <div class="page">
                ...
        </div>
    </t>
</template>
```

El report_name que se indica en la definición del informe (report) sería el de la vista padre y el documento tendría un nombre derivado de este.

Comunitea

# Traducciones (II)

De cara a traducir un módulo, primero debemos instalarlo y luego exportar los términos a traducir, para esto:

1. Tenemos que activar en Odoo el modo desarrollador
2. En /Configuración/Traducciones/”Importar / Exportar”/Exportar traducción ponemos el idioma al que vamos a traducir, formato “.po” y seleccionamos el módulo a traducir.
3. Guardamos el fichero que crea, que ser algo así como código_idioma.po en una carpeta i18n dentro del módulo.
4. Traducimos el fichero con Poedit o con cualquier editor de texto.

Comunitea

# Ejercicios 17

Hacer traducible el informe que hicimos en el ejercicios anterior y traducir todo el módulo.

Comunitea

# Seguridad (I)

Para poder acceder con usuarios que nos sean “admin” a los modelos de Odoo que desarrollamos tenemos que crear un fichero de nombre ir.model.access.csv en la carpeta security del módulo donde le definiremos los acceso.

El formato del fichero es:

"id","name","model_id:id","group_id:id","perm_read","perm_write","perm_create","perm_unlink"
"access_model",modelo","model_nombre_modelo",id_grupo,1,0,0,0

Comunitea

# Seguridad (II)

A parte de los permiso básicos que dan acceso de lectura, escritura, creación y borrado a todo el modelo se puede definir dentro de estos, límites, es decir que cierto grupo sólo pueda ver los registros de un modelo de los cuales es responsable por ejemplo.

Estas reglas de acceso se crean en un xml en la carpeta security y tiene un formato similar a:

```
<record id="id_registro" model="ir.rule">
        <field name="name">Descripción</field>
        <field name="model_id" ref="model_modelo_nombre"/>
        <field name="global" eval="True"/>
        <field name="domain_force">['|',('company_id','=',False),
('company_id','child_of',[user.company_id.id])]</field>
</record>
```

** global True indica que se ejecuta para todos los usuarios, podríamos ponerle un atributo “groups”  y de valor un grupo/s para que sólo afecte a estos.

Comunitea

# Ejercicio 18

Dar los permisos correspondientes a los modelos que se crearon

Crear un grupo de usuarios “Profesor” en base de datos y limitarle el acceso sólo a sus sesiones, no puede ver las de los demás profesores en cambio sin este grupo si verían todo.

Probar los accesos con un usuario que no sea admin

# Gestión de base de datos

En la venta de inicio de sesión de Odoo se dispone de un enlace “Gestionar Bases de datos”, si estuviera oculto para acceder iríamos a: http://servidor_odoo/web/database/manager

Desde esta interfaz podemos hacer un backup de nuestra base de datos y sus adjuntos, podemos duplicarla y borrarla.

También nos permite crear una base de datos nueva o recuperar un backup

Comunitea

# Referencias

Documentación técnica oficial de Odoo:
https://www.odoo.com/documentation/10.0/

Grupo de la comunidad española, resolución de dudas (altruista):
https://groups.google.com/forum/#!forum/openerp-spain-users

Repositorio de la Comunidad bajo OCA
https://github.com/OCA

Foro Odoo Spain (Poco uso)
https://odoospain.odoo.com/forum/ayuda-1

Foro oficial Odoo
https://www.odoo.com/es_ES/forum/ayuda-1