MÓDULO DE HELPDESK

Por: Iago Arthaud Meije

Módulo Helpdesk

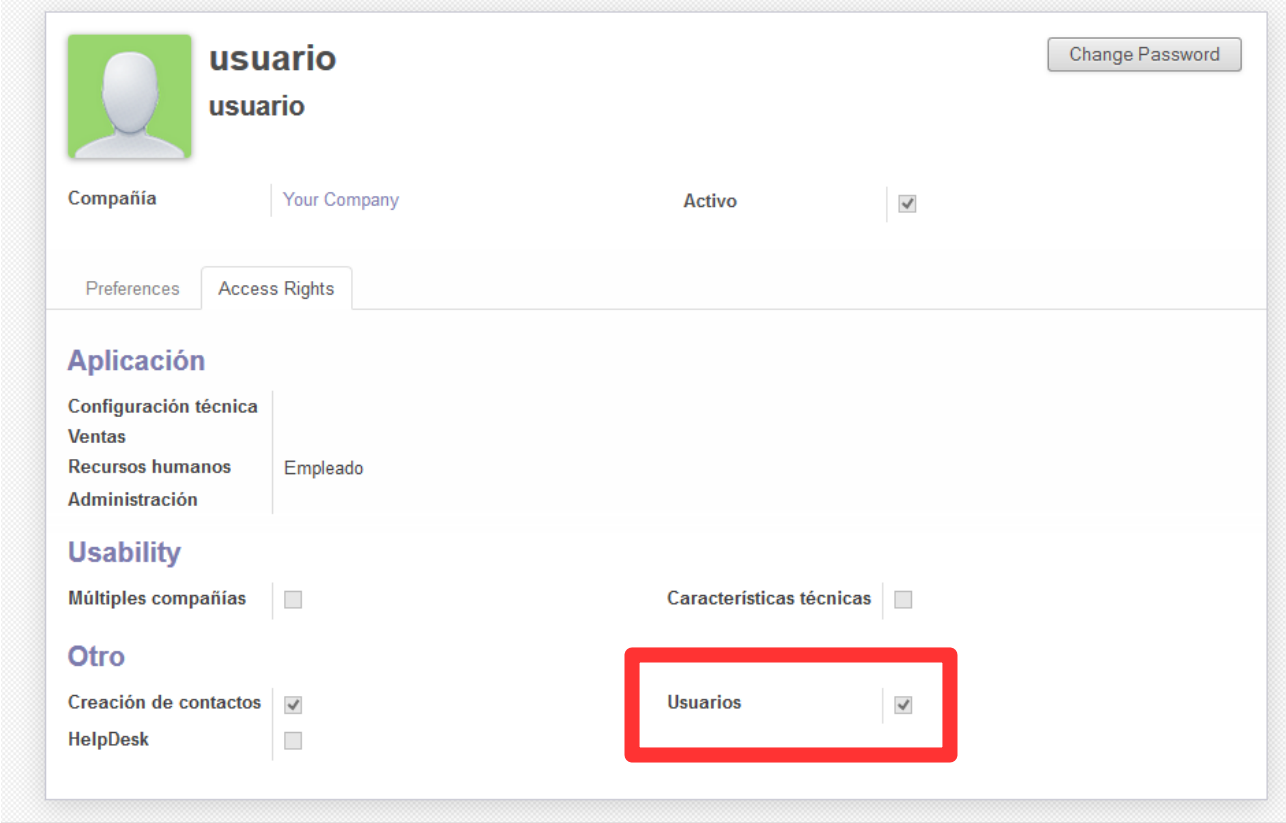
Introducción

El módulo que he creado está diseñador para ser usado por el departamento de helpdesk. Basicamente tiene la funcionalidad de poder abrir o cerrar incidencias (tickets). Ésta incidencias tienen una descripcion de las mismas, la fecha, el usuario que la abríó, estado, etc...
Éstas incidencias o tickets pueden ser abiertas por un grupo de usuarios llamado **"usuarios"**, que se tienen que asignar en la pestaña de **"Acces Rights"** de la configuración de usuario. Éste grupo de usuarios solo podra crear tickets con determinadas restricciones, y nunca podra eliminar, modificar tickets ya creados.
Además de este grupo tenesmo el otro llamado **"helpdesk"**, el cual tiene menos restricciones que el último, ya que son los encargado de modificar las incidencias, para poder registrarlas como cerradas una vez son resueltas, también son capaces de crear incidencias.

Funcionamiento

Usuario Crear incidencias

Antes de nada asignamos a un usuario permisos para poder crear la incidencia.



Una vez hecho podremos acceder al menú principal del módulo, nos dirigimos a "Añadir Incidencias" para abrir una.

De primeras podemos comprobar varias cosas, la primera es que el campo "No. Ticket" viene con una numeración por defecto (tiene una funcion sequence de Openerp), este campo, el estado y el usuario estan en modo lectura, por lo que no se pueden cambiar. Además vienen con unos valores predeterminados, que son el usuario que lo abrió y el estado que siempre es abierto al crear. Por último la fecha viene tambien actualizada a fecha de hoy, pero este campo si que se podría cambiar.

Tambíen podemos fijarnos en que todos los campos son obligatorios, por lo que no se puede dejar uno sin rellenar.

Como vemos nos resalta el campo sin rellenar en rojo y nos muestra un mensaje avisandonos de que lo rellenemos.
Ahora crearemos la incidencia y podremos ir al menu "Ver Incidencias" para comprobar que ha sido creada.

Éste menú tiene tres vistas, en tree, en search y en form, si clicamos en uno de los tickets, podemos pasar a la vista form, y comprobar que el usuario no tiene permisos ni para modificar ni para borrar los tickets.

También podemos apreciar que en esta vista aparecen tres campos, que en la de crear no. Ésto es así porque, aunque en el modulo python, le tenga restringido los permisos de escritura a este grupo de usuarios, si no los ocultara, seguirían apareciendo, aunque lo que escribieramos en ellos no se guardaría, pero daria lugar a confusiones, y demas, es mejor que el usuario normal no pueda hacer nada mas que los estrictamente necesario.

Helpdesk modifica incidencias

Ahora con el "usuario2" que está en el grupo de "helpdesk" procederemos a ver la lista de incidencias, antes creamos una incidencia con este mismo usuario para comprobar que podemos ver todas, independientemente de quien la creó, al final volveremos a loguear con el primer usuario, para comprobar que solo puede ver las incidencias que el mismo abrió.

Antes de modificar la incidencia podemos ver como funcionan los filtros de la vista search, si clicamos en el desplegable del buscador podemos seleccionar uno de los filtros creados predeterminadamente, que son filtrar por tickets cerrados, o por abiertos.

También podemos buscar escribiendo en el campo, y elegir por que campo buscar (todos los campos menos las descripciones pueden ser buscados)

Ahora si seleccionamos la incidencia que queremos resolver, nos cambiará a la vista form, y veremos que aparece un botón que nos permite editarla para resolverla.

Presionamos el botón para poder editar los campos. Tal como están collocados los campos, y sabiendo que los helpdesk solo necesitan cambiar los campos estado, solucionado por , fecha resolucion y descripcion resolucion, lo normal seria empezar a rellenarlo de arriba a abajo empezando por estado. Es por eso que éste último campo tiene una funcion on_change, que ya nos rellena los campos solucionado por y fecha resolucion, con nuestro usuario logueado y la fecha de hoy.

Guardamos, y ya quedaría todo registrado. Si volvemos a loguear con el primer usuario, podemos comprobar como la incidencia ya está cerrada, y la que abrió el usuario2 no está visible para el usuario1.

Explicación Código

__openerp__.py

```
'data': [
        'res_helpdesk_view.xml',
                'security/security.xml',
                'security/ir.model.access.csv'
    ],
```

Lo más importante aquí para explicar son estas lineas. En ellas le decimos que vamos a usar éstos documentos, que son de arriba a abajo, el xml con las vistas, el xml de security, en el que creamos las reglas y los grupos que acceden al módulo, y la ruta del csv, donde están los permisos de acceso que le otorgamos a éstos grupos.
Éste orden es importante, puesto que si ponemos el xml de security después del csv, este último va a intentar dar permisos a unos grupos que aún no han sido creados y el openerp se volverá loco.

security.xml

En este xml como dije antes, creamos primero los grupos ejemplo:

```
<record id="all_users" model="res.groups">
        <field name="name">Usuarios</field>
    </record>
```

Siendo, el id, el identificador que usaremos para referirnos al grupo, el model, el modelo en el que lo creamos, y dentro de la etiqueta name, el texto que mostrará el grupo (el nombre que se verá).
Luego tenemos la regla:

```
<record id="rule_own" model="ir.rule">
        <field name="name">only_own_tickets</field>
        <field name="model_id" ref="model_res_helpdesk" />
        <field name="groups" eval="[(4, ref('all_users'))]"/>
        <field name="domain_force">[('usuario.id','=',user.id)]</field>
                <field name="perm_create" eval="True"/>
                <field name="perm_unlink" eval="False"/>
                <field name="perm_write" eval="False"/>
                <field name="perm_read" eval="True"/>
    </record>
```

En la que podemos remarcar que "ref" hace referencia al modelo que se le va a aplicar, la línea que tiene "groups" eval="[(4, ref('all_users'))]" indica que se va a aplicar a determinados grupos, y lo que va entre [] indica que se le va a aplicar al grupo all_users.
Luego tenemos la linea que fuerza e dominio, que lo que hace es que solo muestre los objetos que tengan en el campo usuario, un usuario con id igual al que intenta acceder, osea que el usuario solo pueda ver sus propios tickets.
Las últimas lineas indican un segundo nivel de permisos aparte del del csv, como queremos que tengan los mismos, se los volvemos a poner (True indica que tiene y false que no), porque si no pondria todo a permitido.

ir.model.access.csv

id,name,model_id:id,group_id:id,perm_read,perm_write,perm_create,perm_unlink
accesolectura,lectura,model_res_helpdesk,all_users,1,0,1,0
accesoescritura,escritura,model_res_helpdesk,helpdesk,1,1,1,1

Éste es el contenido del csv, la primera línea es la cabezera de los campos, y las siguientes, son los permisos de acceso, que van delimitando cada campo por una ",".
tenemos el id por el que referirse a la acl, el nombre que le damos, el modelo al que se aplica, y el grupo de usuario al que se aplica. Los últimos campos son los permisos en si, "1" significa que tiene permisos y "0" que no.

res_helpdesk.py

```
        'usuario_sol': fields.many2one(
                'res.users',
                'Solucionado por: ',
                read=['helpdesk.all_users','helpdesk.helpdesk','base.group_system'],
                write=['helpdesk.helpdesk','base.group_system']),
        'fecha':fields.date('Fecha', required=True),
        'fecha_resol': fields.date('Fecha Resolución',
                read=['helpdesk.all_users','helpdesk.helpdesk','base.group_system'],
                write=['helpdesk.helpdesk','base.group_system']),
        'estado':fields.selection((('a','Abierto'), ('c','Cerrado')),('Estado'), required=True)
```

Lo más destacable del módulo python es el uso de los campos many2one para asociar los tickets a los usuarios o el hecho de restringir algunos campos dando permisos de escritura o lectura (si en un campo le pones permisos, como write=['helpdesk.helpdesk','base.group_system'] debes de indicarle también el de lectura, pues no lo pone implícitamente) a determinados grupos ya a nivel de módulo(también se puede a nivel de vista). Además hay un campo de tipo selection.

```
_defaults = {
        'estado' : 'a',
        'usuario': lambda obj, cr, uid, context: uid,
        'fecha': fields.date.today(),
        'name':lambda obj, cr, uid, context: obj.pool.get('ir.sequence').get(cr, uid, 'res.helpdesk.ticket'),
    }
```

en _defaults podemos poner los valores por defecto al crear un ticket en determinados campos. Obviamente no queremos que los tickets se abran cerrados por lo que ponemos el campo a abierto por defecto, rellenamos el campo usuario con el usuario logueado, ponemos la fecha actual y aplicamos al campo name una secuencia que tenemos definida en el xml que luego explicaré('res.helpdesk.ticket' es el id de la secuencia).

```
def onchange_estado(self, cr, uid, ids, context=None):
        vals = {
            'usuario_sol': uid,
            'fecha_resol': fields.date.today(),
        }
        return {'value': vals}
```

Por último tenemos una funcion que saltara con el on_change que nos rellena el usuario que resuelve la incidencia y la fecha actual.

res_helpdesk_view.xml

Por último voy a describir algunas cosas del xml.
Tenemos dos view Form, una que es para ver mas detalladamente las incidencias, o para el grupo helpdesk modificarlas, y otro para crearlas unicamente. En la form que no queremos que se puedan crear, lo que hice fue deshabilitar el botón de crear, para que no aparezca poniendo a false el create, esta línea:

```
<form create="false" string="helpdesk">
```

Además como en esta misma vista es en la que los helpdesk van o modificar los tickets, añadimos la funcion onchange al campo estado:

```
<field name="estado" on_change="onchange_estado()"/>
```

Para poder saber que vistas va a utilizar cada menú tenemos que indicárselo (la primera vista definida de cada tipo es la por defecto, en este caso la vista de ver/modificar, para las otras tenemos que especificarselo) en el menu action. En este caso tenemos dos, el que coje la vista por defecto (ver/modificar) y la que coje la vista de solo añadir:

```
<record model="ir.actions.act_window" id="action_form_add_helpdesk">
                <field name="name">Helpdesk</field>
                <field name="res_model">res.helpdesk</field>
                <field name="view_type">form</field>
                <field name="view_mode">form</field>
                <field name="view_id" ref="res_helpdesk_form_add"/>
        </record>
```

Con la última linea le indicamos que view cojer, de paso le indicamos que tenga otras vistas, como tree en el view_mode. En un menú decimos que use este action menu y en el otro, el otro action que usa las vistas por defecto.
Por último de todo tenemos la sequence de Openerp:

```
        <record id="seq_type_res_helpdesk_ticket" model="ir.sequence.type">
            <field name="name">Helpdesk Ticket</field>
            <field name="code">res.helpdesk.ticket</field>
        </record>
        <record id="seq_res_helpdesk_ticket" model="ir.sequence">
            <field name="name">Helpdesk Ticket</field>
            <field name="code">res.helpdesk.ticket</field>
            <field name="prefix">TK-</field>
            <field name="padding">6</field>
            <field name="number_increment">1</field>
        </record>
```

El primer record es para declarar el tipo, que es ir.sequence.type y el segundo declarar la sequence en si. Donde le indicamos el modelo ir.sequence (porque es una sequence) el prefijo que tendrá el char (TK- en este caso, tambien puede tener un sufijo), el número de ceros a la izquierda por defecto (000001), y cuanto se incrementará cada vez que sea llamado (1 en este caso).
Ésto daría resultado a que cada vez valla incrementandose este campo (TK-000001,TK-000002,TK-000003...). Es importante saber que el campo que sea incrementado con ésta secuencia, o con cualquier otro tipo, tenga prefijo, sufijo o no, debe ser un campo char/text, en los campos numericos como integer no funcionaría.