Práctica 3: Primeros módulos Odoo

Indice

0. CONTEXTO Y PASOS PREVIOS

En este documento explicare de forma sencilla los pasos que he seguido para crear y mejorar mis primeros módulos en Odoo.

Además de describir los comandos que he utilizado, quiero dejar por escrito el contexto de trabajo: estoy usando una instalación de Odoo levantada con *docker compose*, con un contenedor para la aplicación web y otro para la base de datos. El objetivo principal de esta práctica es entender la estructura

básica de un módulo, cómo se instala desde la interfaz de Odoo y cómo se pueden hacer pequeñas mejoras sobre un módulo ya generado.

-Pasos previos

-Activar el modo de desarrollador

para activar el modo desarrollador nos deberemos ir a los ajustes de odoo ->Luego haremos click en Activar Modo Desarrollador.

Gracias a este modo, se habilitan opciones adicionales en el menú, como la posibilidad de ver la información técnica de los registros, depurar vistas, acceder al listado de módulos instalados y recargar la lista de aplicaciones después de añadir un nuevo módulo en la carpeta de addons.

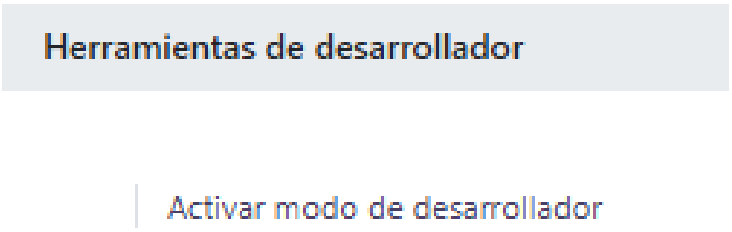

Figura 1: Imagen de la opción

1. MODULO HOLA MUNDO

Lo primero que he hecho ha sido entrar en mi carpeta de odoo (entregada en la primera practica) y creado dentro de la carpeta de addons (en su día di de alta con el docker_compose), la carpeta "holamundo".

Dentro de esta carpeta, he metido 2 archivos. Un archivo con el nombre : `__init__.py` y otro con el nombre `__manifest__.py`.

El módulo Hola Mundo me sirve como prueba mínima para comprobar que Odoo detecta correctamente los módulos personalizados que añado en la ruta de addons. Es un ejemplo sencillo pero muy útil para verificar que la configuración del entorno con Docker está bien hecha y que los permisos de las carpetas permiten que Odoo lea el contenido del módulo sin errores.

Dentro del manifest he insertado el nombre que tendrá. Para ello, he insertado dentro del archivo lo siguiente:

```
{
```

```
    'name': 'Ejemplo Hola mundo',
}
```

El fichero `__init__.py` se utiliza para indicar a Odoo qué paquetes de Python se deben cargar al iniciar el módulo. En módulos más complejos, desde este archivo se suele importar la carpeta `models` y otros submódulos. El archivo `__manifest__.py` contiene la información básica del módulo: nombre, versión, dependencias, datos que se cargan, etc. En mi caso solo he definido el campo `'name'`, pero en un manifest real también se podrían añadir otros campos como `'version'`, `'depends'` o `'data'`.

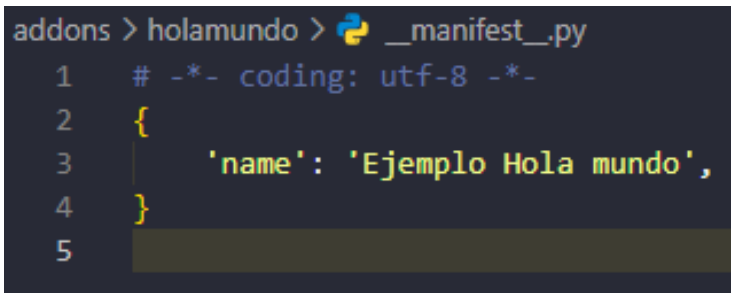

Figura 2: Imagen del código

Luego, para forzar la recarga del módulo. Debemos reiniciar el contenedor:

```
docker compose restart odoo
```

Luego , tras abrir odoo en nuestro navegador, podremos visionar el modulo para instalar

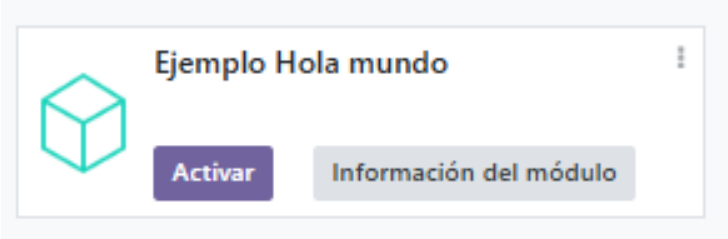

Figura 3: Imagen del módulo

`Problemas ocasionados:`

```
Para que el módulo apareciese, tumbe y levante el contenedor de odoo.
Debido a un problema en mi código del ".yaml", el contenedor se levantó sobreesc
A partir de ahora, es suficiente con reiniciar el contenedor web.
docker compose restart web
```

Este problema me ha servido para entender mejor la diferencia entre reiniciar un contenedor y recrearlo desde cero. Al recrearlo, perdí la base de datos anterior porque no estaba usando correctamente los volúmenes de Docker. A partir de este error he aprendido que, cuando solo quiero que Odoo vuelva a detectar cambios en el código de un módulo, basta con hacer un `docker compose restart` del servicio correspondiente, sin borrar ni recrear toda la configuración.

Resumen de funcionamiento: este módulo de Hola Mundo no hace nada especial en Odoo, solo aparece en la lista de módulos cuando actualizamos las aplicaciones en modo desarrollador y se puede instalar. Nos sirve para comprobar que la carpeta del módulo y los archivos `__init__.py` y `__manifest__.py` están bien creados y que Odoo detecta correctamente los módulos que añadimos.

2. MODULO LISTA DE TAREAS

Con el modulo de desarrollador activado, ejecutaremos el scaffold dentro el contenedor web. Para ello ejecutamos:

```
docker compose exec web odoo scaffold lista_tareas /mnt/extra-addons
```

Si lo hicimos bien, se nos habrá creado la estructura de la carpeta del módulo de listatareas.

Ahora haremos un paso opcional pero que hará que el proceso de los módulos sea mas cómodo (En windows):

```
docker compose exec web bash -lc "chmod -R 777 /mnt/extra-addons/lista_tareas"
```

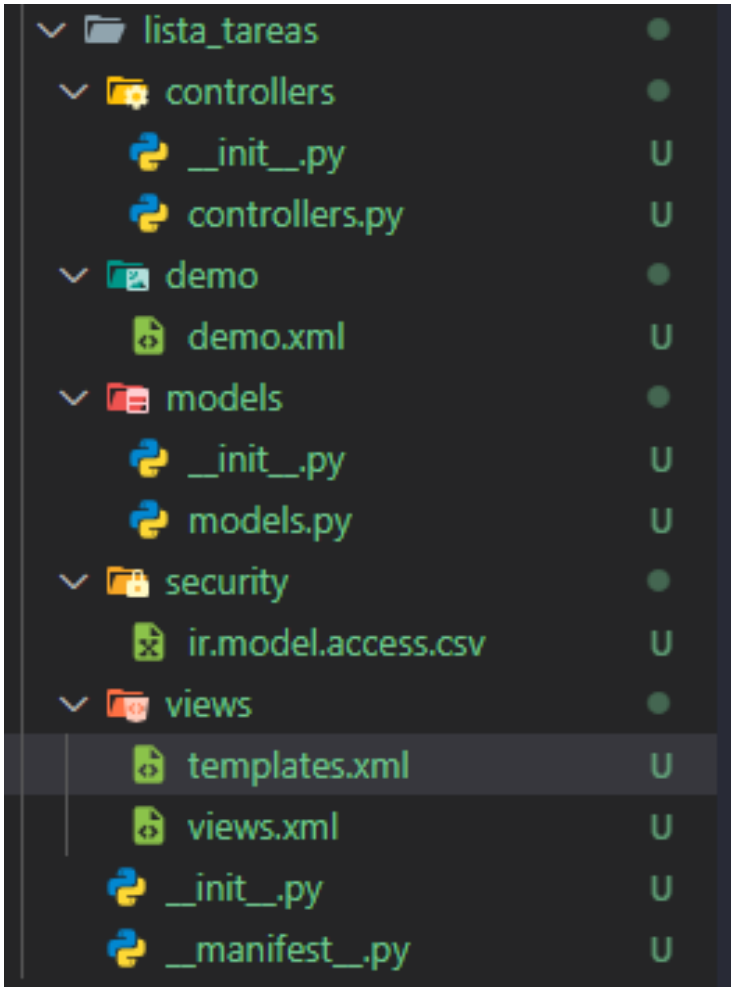

Figura 4: Imagen de las carpetas del módulo

Una vez hecho esto, con la estructura montada, rellenaremos los ficheros del módulo. Dentro de /addons/`lista_tareas` rellenaremos los archivos tal cual nos dice el ejemplo de la documentación otorgada.

En este módulo se define un modelo sencillo de lista de tareas. Entre los campos principales que se pueden definir están: el nombre de la tarea, la prioridad, un campo booleano para marcar si está realizada y otros campos auxiliares que ayudan a organizar el trabajo. Además, se crean vistas de lista y formulario para poder gestionar las tareas desde la interfaz web de Odoo.

Los ficheros más importantes que genera el comando `scaffold` para este módulo son:

- `models/lista_tareas.py`: donde se define la clase del modelo y sus campos.
- `views/lista_tareas_views.xml`: donde se definen las vistas de lista y de formulario.
- `__manifest__.py`: donde se declara el módulo y los datos que se van a cargar.
- `security/ir.model.access.csv`: donde se configuran los permisos de acceso al modelo.

Gracias a este módulo he podido practicar cómo se relaciona la parte de Python (modelo de datos) con la parte de XML (vistas) y con los permisos de seguridad, que son necesarios para que el usuario pueda ver y editar los registros desde Odoo.

Resumen de funcionamiento: este módulo de lista de tareas ya es un módulo "de verdad" creado con `odoo scaffold`. Ese comando le dice a Odoo que nos prepare toda la estructura básica de un módulo: el `__manifest__.py` para describir el módulo, el `__init__.py` para cargar la lógica, la carpeta `models` para el modelo de datos, la carpeta `views` para las vistas y menús, y la parte de `security` para los permisos. A partir de esa base, el módulo añade un modelo nuevo de tareas en `models/lista_tareas.py`, unas vistas y menús en `views/lista_tareas_views.xml` y sus reglas de acceso en `security/ir.model.access.csv`, de forma que las tareas se pueden crear, ver y editar dentro de Odoo como una aplicación más.

3.Realizar un mejora en el modulo.

He pensado en que la lista de tareas, al ser compartida, estaría bien que tuviese un campo de responsable. De esta manera, cada uno hace el seguimiento oportuno de sus tareas asignadas. Para hacerlo, he añadido dentro en views.xml un simple <field name = responsable/>con el mismo formato que el resto de fields. El resto lo he dejado igual.

Luego, simplemente he añadido a la clase de ListaTareas : responsable = fields.Char(string=Responsable"). Tras esto, he modificado el modulo de forma que ahora, pueda asignar un responsable:

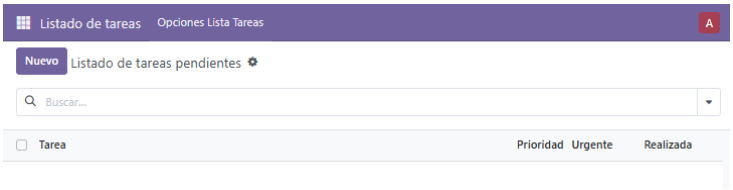

Figura 5: Campo responsable

Además de añadir el campo, he comprobado que aparece correctamente tanto en la vista de lista como en la vista de formulario y que se guarda en la base de datos al crear o editar una tarea. Esta mejora, aunque es pequeña, demuestra cómo a partir de un módulo generado con `scaffold` se pueden ir incorporando nuevos campos y adaptando el módulo a las necesidades reales de los usuarios.

También he podido comprobar cómo al actualizar el código del módulo es necesario actualizar la aplicación desde el menú de Odoo para que se apliquen los cambios en las vistas y en el modelo de datos. Esto forma parte del ciclo normal de desarrollo: modificar código, actualizar el módulo y probar de nuevo en la interfaz.

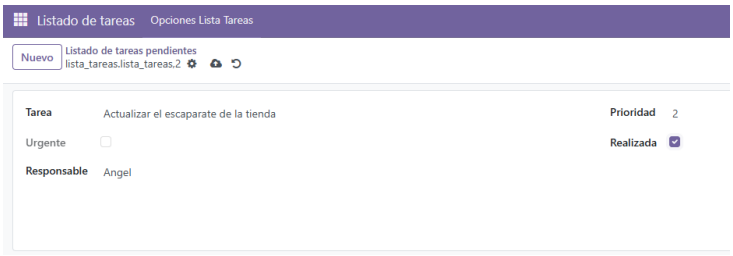

Figura 6: Campo responsable

Resumen de funcionamiento: en esta mejora no creo un módulo nuevo, sino que aprovecho la estructura que generó `odoo scaffold` y la amplío un poco. Por un lado, añado el campo `responsable` al modelo en

`models/lista_tareas.py`, y por otro lado lo muestro en las vistas editando el `views.xml`. Con esto, el módulo sigue funcionando igual que antes, pero ahora cada tarea guarda también quién es el responsable y esa información viaja desde la base de datos a la pantalla (y al revés) usando los mismos ficheros del módulo.