

UD05. ENTORNO DE DESARROLLO

Sistemas de Gestión Empresarial

2 Curso // CFGS DAM // Informática y Comunicaciones

Alfredo Oltra

**Cicles
Formatius**

ÍNDIX

1 INTRODUCCIÓN	4
1.1 ¿Cómo vamos a orientar esta unidad?	4
2 DIRECTORIO DE TRABAJO	4
2.1 Instalación manual	4
2.2 Instalación mediante <i>Docker</i> y <i>Docker Compose</i>	5
2.3 Instalación mediante <i>Odoodock</i>	5
3 ENTORNO DE DESARROLLO	6
3.1 <i>Visual Studio Code</i>	6
3.1.1 Control de versiones usando <i>git</i> + <i>Visual Studio Code</i> .	6
3.1.2 Trabajando desde <i>Odoodock</i>	6
3.2 <i>PyCharm</i>	7
3.2.1 Control de versiones usando <i>git</i> + <i>PyCharm</i>	7
4 MODO DESARROLLADOR	7
5 PRIMER MÓDULO: <i>HOLA MUNDO</i>	8
6 CREANDO MÓDULOS	9
6.1 Scaffolding	9
6.2 Scaffolding	10
6.3 Actualización de módulos	10
7 FICHERO <i>__manifest__.py</i>	11
8 EJEMPLO COMPLETO: <i>LISTA DE TAREAS</i>	14
9 BIBLIOGRAFIA	17
10 AUTORES	17

Versión: 231127.1150


Licencia




Reconocimiento – NoComercial – CompartirIgual (by-nc-sa). No se permite un uso comercial de la obra original ni de las posibles obras derivadas, la distribución de las cuales se debe hacer con una licencia igual a la que regula la obra original.

Nomenclatura

A lo largo de este tema se utilizarán distintos símbolos para distinguir elementos importantes dentro del contenido. Estos símbolos son:

 **Atención.** Importante prestar atención a esta información.

 **Interesante.** Ofrece información sobre algún detalle a tener en cuenta.

1 INTRODUCCIÓN

Vamos a empezar a desarrollar módulos y lo primero es instalar y conocer la infraestructura con la que vamos a trabajar. En la unidad 2 se explicaba cómo instalar *Odoo* para el trabajo en desarrollo, pero en esta unidad vamos a profundizar un poco más incluyendo tanto el propio *Odoo*, como el IDE de desarrollo con sus respectivos plugins. Con todo esto listo, realizaremos nuestros primeros módulos de *Odoo*.

1.1 ¿Cómo vamos a orientar esta unidad?

En primer lugar, preparamos todo el entorno de desarrollo. Con un sistema ERP *Odoo* instalado, se pueden desarrollar módulos que amplían su funcionalidad sin una gran preparación del entorno. Tan sólo una terminal y un editor de texto son suficientes para poder desarrollar con *Odoo*.

No obstante, se recomienda realizar una configuración previa tanto del propio sistema ERP *Odoo* como de las herramientas de desarrollo asociadas para hacer el trabajo más fácil y aumentar la productividad.

Una vez configurado el entorno de desarrollo, comenzaremos a introducir los primeros módulos de *Odoo*. Comenzaremos con un sencillo módulo “que no hace nada”, pero que ya se puede instalar. Este módulo nos ayudará a validar si tenemos una configuración correcta de nuestro entorno de desarrollo.

El desarrollo de módulos de *Odoo* puede llegar a ser muy complejo y sólo los programadores expertos son capaces de profundizar desde el principio sin ver cómo va funcionando. En la siguiente unidad ya profundizaremos en mayor detalle en la programación de módulos para *Odoo*.

2 DIRECTORIO DE TRABAJO

Como ya vimos en la unidad 2, es importante tener un directorio para incluir y desarrollar módulos en *Odoo*. En este apartado comentaremos cómo poner en marcha este directorio tanto en una instalación manual como utilizando *Docker* u *Odoodock*.

2.1 Instalación manual

Si hemos hecho una instalación manual tal como se explicó en la unidad 2, una secuencia de comandos para conseguir crear este directorio en la carpeta `/var/lib/odoo/modules` es:

```
# Estando situado en /var/lib/odoo:
# Creamos directorio modules
mkdir modules
# Creamos un módulo llamado prueba con la utilidad odoo scaffold
odoo scaffold prueba ./modules

# Modificamos el path de los addons.
odoo --addons-path="/var/lib/odoo/modules,/usr/lib/python3/dist-packages/
odoo/addons" -save
```

```
# Lanzamos el servidor Odoo y actualizamos el módulo prueba en la bd empresa  
odoo -u pruebas -d empresa
```

Con esta acción, lo que conseguimos es añadir nuestro directorio *modules* al PATH de *Odoo* en el archivo de configuración *.odoorc* propio del usuario *odoo*. Con el último comando arrancamos *Odoo* actualizando este módulo (*prueba*) en la base de datos *empresa*.



Más adelante veremos el uso del comando *scaffold*, por ahora simplemente piensa que su utilidad es crear la estructura de un módulo.



Los módulos deben poder ser al menos leídos por el usuario *odoo*, que es el que lanza el servicio. En desarrollo es habitual que ese usuario además de leer tenga permisos de escritura para poder realizar cambios.

2.2 Instalación mediante Docker y Docker Compose

En caso de utilizar el fichero *docker-compose.yml* básico definido en la unidad 2, el directorio configurado por defecto es */mnt/extra-addons*. Ese directorio es mapeado en la máquina anfitriona como *volumesOdoo/addons*

2.3 Instalación mediante Odoodock

En el caso de *Odoodock*, la carpeta del contenedor donde están los módulos es */mnt/extra-addons*. Por defecto, esa carpeta es mapeada en la máquina virtual a una carpeta llamada *addons* situada al mismo nivel que *odoodock*. Es posible cambiar su ubicación configurando modificando la variable *APP_MODULE_PATH_HOST* del fichero *.env*.

3 ENTORNO DE DESARROLLO

Existen diversos entornos de desarrollo que permiten desarrollar módulos para *Odoo*. Los más utilizados son *Visual Studio Code* y *PyCharm*. Además es importante utilizar algunas herramientas adicionales, como *git* junto con plataformas como *github* o *gitlab*.

3.1 Visual Studio Code

Visual Studio Code es la opción recomendada. Es muy potente y posee un gran ecosistema de plugins para ampliar su funcionalidad. Pensando en el desarrollo de *Odoo* es recomendable tener instaladas las siguientes extensiones:

- *Python*: soporte para el lenguaje *Python*, incluyendo características como *IntelliSense*, *linting*, depuración, navegación de código, formateo de código, refactorización, explorador de variables o explorador de pruebas.
- *Odoo-snippets*: proporciona fragmentos de código (*snippets*) predefinidos tanto para *python*, *javascript*, *xml* y *csv*
- *Excel viewer*: proporciona editores personalizados y vistas previas para archivos CSV y hojas de cálculo Excel.

Además pensando en su uso con *Odoodock*, son interesantes:

- *Docker*: facilita la creación, gestión e implantación de aplicaciones en contenedores.
- *Dev Containers*: permite usar un contenedor como un entorno de desarrollo completo.

3.1.1 Control de versiones usando *git* + *Visual Studio Code*.

Visual Studio Code incluye de serie una extensión para la gestión de repositorios *git*. Sus características son suficientes para el desarrollo del trabajo que vamos a realizar en el curso, pero, en entornos más profesionales es más recomendable optar por una herramienta externa como *GitKraken* o *SourceTree* (por nombrar algunas).



Hay más información sobre el uso de *git* en *Visual Studio Code* en [este enlace](#).

3.1.2 Trabajando desde *Odoodock*

Aunque es posible trabajar con cualquier IDE, *Odoodock* está pensando para trabajar con *Visual Studio Code* y, especialmente desde dentro del contenedor. Para ello es necesaria la extensión *Dev Containers*. Esto hace que el proceso de arranque sea un poco diferente al habitual ya que es necesario acceder al contenedor desde la extensión y adjuntarlo a una instancia de *Visual Studio Code*.



En este caso, es posible que en esa instancia interna al contenedor pueda no estar instaladas las extensiones que existen en local, por lo que habría que volver a instalarlas.



Hay más información sobre el uso de *Odoodock* con *Visual Studio Code* en esta [lista de videos](#) y en la [documentación](#).

3.2 PyCharm

Otro editor muy recomendado en *Pycharm*. Por defecto, no reconoce los elementos del framework *Odoo*, pero sí los del lenguaje *Python*. Si queremos facilitar el desarrollo podemos instalar la extensión de *Odoo Pycharm Templates*.

3.2.1 Control de versiones usando *git* + *PyCharm*

PyCharm también permite el uso de un sistema de control de versiones *git* desde su entorno.



Hay más información sobre el uso de *git* en *PyCharm* en [este enlace](#).

4 MODO DESARROLLADOR

Una de las herramientas más interesante que nos proporciona *Odoo* a la hora de desarrollar es activar el modo desarrollador. Existen varias opciones para hacerlo, entre las que se encuentran:

- *Desde los ajustes*. En la parte final de la página de ajustes existe la opción de poder activar el modo desarrollador. Para ello es necesario tener como mínimo un módulo instalado.
- *A través de una extensión*. Es la opción recomendada, ya que permite el cambio entre modos de una manera sencilla. Puedes encontrarlas en estos enlaces para [Firefox](#) y [Chrome](#).
- *A través de la URL*. Agregando a la URL de conexión `?debug=true` o `?debug=1`. Para desactivarlo simplemente hay que eliminarlo o cambiarlo por `?debug=0`.



Es posible encontrar más información sobre la activación del modo debug en [este enlace](#).

5 PRIMER MÓDULO: HOLA MUNDO

Como es procedente en el mundo de la programación vamos a empezar a desarrollar módulos creando el módulo más sencillo que hay, el *Hola Mundo*.

En este caso el *Hola Mundo* va a consistir en un módulo básico que no va a hacer nada, únicamente aparecer en el lista de módulos para ser instalado.



La creación de este módulo tiene un fin completamente didáctico y nos ayudará a comprobar que nuestro sistema está configurado correctamente para poder detectar y utilizar los módulos que desarrollemos.

Dentro del contexto del desarrollo de módulos, un módulo no es ni más ni menos que una carpeta que contiene dentro, como mínimo, dos ficheros Python: `__init__.py` y `__manifest__.py`

El nombre de la carpeta tiene su importancia, ya que va a ser considerado el nombre técnico del módulo. Ese nombre técnico va a aparecer, por ejemplo, como prefijo en las tablas de la base de datos o como nombres de los modelos. En principio, ese nombre técnico puede ser cualquiera, aunque teniendo en cuenta las repercusiones que va a tener a lo largo de todo el código, ciertos nombres suelen ser problemáticos. Lo aconsejable es utilizar la nomenclatura *snake_case* (y no usar guiones – ni empezar con números).

Por lo que respecta a los ficheros:

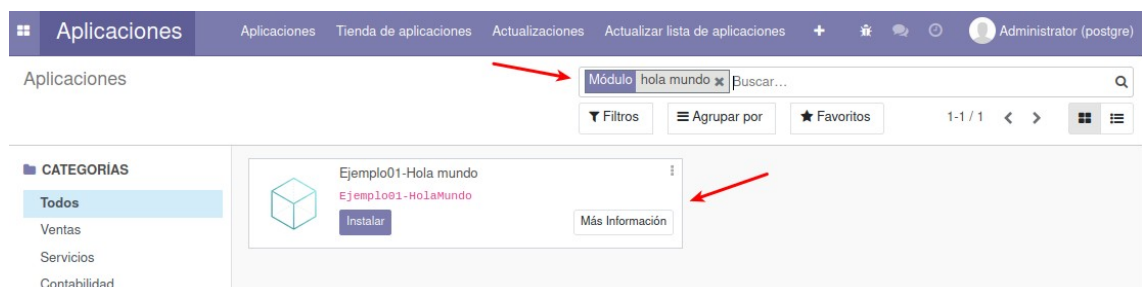
- Como ya sabemos la función del fichero `__init__.py` es indicarle al intérprete que una carpeta es un módulo y que, en muchas ocasiones, suele estar vacío.
- Por lo que respecta al `__manifest__.py` es el fichero que realmente declara el módulo. Su contenido es un diccionario (en formato *json*), que incluye los metadatos. Más adelante hablaremos de él, pero para un módulo mínimo lo único necesario es la declaración del nombre del módulo.

```
# -*- coding: utf-8 -*-
{'name': 'Hola mundo'}
```

Una vez creada la estructura, con el *Modo Desarrollador* activado, podréis ir al listado de módulos y *Actualizar la lista de aplicaciones* tal como se observa en la imagen



Tras ello, eliminando los filtros de búsqueda por defecto y buscando *hola mundo*, podremos encontrar nuestro módulo. Si todo ha funcionado correctamente, veremos algo similar a:



Ahora podremos instalar nuestro módulo para probarlo (aunque este ejemplo no hace nada).

6 CREANDO MÓDULOS

6.1 Scaffolding

Evidentemente un modulo más profesional contiene en su interior una estructura más compleja que la del módulo anterior. Aunque eso no implica que no pueda crearse a mano, lo habitual es utilizar una opción de *scaffolding* que permite generar la estructura de manera automática.



Otra opción es el uso de una plantilla de estructura de módulo que se puede descargar desde la página de Odoo.

Para ello:

- desde una instalación nativa:

```
$ odoo scaffold mi_modulo /mnt/extra-addons/
```

- desde un contenedor docker

```
$ docker-compose exec web /bin/bash
> odoo scaffold mi_modulo /mnt/extra-addons/
```

- desde *Odoodock*

Las opciones son varias, pero la más recomendable es el uso del script de creación de módulos *create-module.sh*. Así, desde la carpeta *odoodock*:

```
$ ./create-module.sh -s mi_modulo
```



Puedes ver el resto de opciones desde la [documentación de Odoodock](#).

Si todo ha ido bien, dentro de la carpeta */mnt/extra-addons*, se habrá creado una carpeta *mi_modulo*.

Si se está dentro de un contenedor Docker, es recomendable darle permisos completos para poder editar fácilmente fuera del contenedor. Para ello desde dentro del contenedor podemos cambiar los permisos:

```
chmod 777 -R /mnt/extra-addons/mi_modulo
```



En caso de usar *Odoodock*, no es conveniente cambiar los permisos, ya que el trabajo de desarrollo se realiza desde su interior.

Al igual que antes, actualizando el listado de módulos, quitamos filtros y buscamos *mi_modulo*, y podremos acceder a él.

El módulo creado contiene un código de ejemplo, pero por defecto todo ese código está comentado. Si queremos habilitarlo, debemos descomentar el contenido de todos los ficheros creados.

6.2 Scaffolding

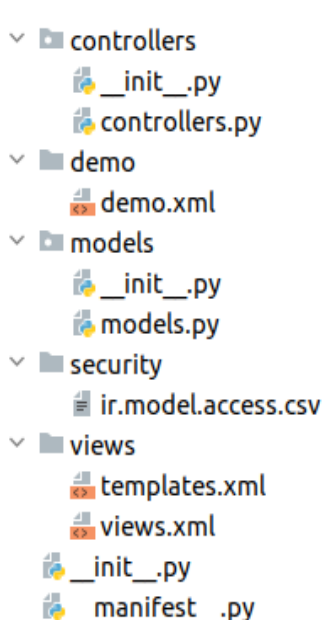
Como en cualquier framework, los directorios donde se programan tienen una estructura determinada con unos ficheros y con unos nombres y extensiones determinados.

En el caso de *Odoo*, todo empieza con un fichero *Python* llamado `__manifest__.py` que contiene la información necesaria para interpretar todos los ficheros que contiene el directorio. Esta información está almacenada usando una estructura diccionario de *Python*.

Además, como en cualquier paquete de *Python*, el directorio contiene un fichero `__init__.py`. Este tiene el nombre de los ficheros *Python* o directorios que contienen la lógica del módulo.

Los subdirectorios con ficheros *Python* de la estructura creada con *odoo scaffold* también tendrán su propio fichero `__init__.py`.

Internamente, la carpeta creada con *odoo scaffold* tiene el contenido que se puede ver en la imagen



En él:

- *models/models.py*: define un ejemplo del modelo de datos y sus campos.
- *views/views.xml*: describe las vistas de nuestro módulo (formulario, árbol, menús, etc.).
- *demo/demo.xml*: incluye datos *demo* para el ejemplo propuesto de modelo.
- *controllers/controllers.py*: contiene un ejemplo de controlador de rutas, implementando algunas rutas.
- *views/templates.xml*: contiene dos ejemplos de vistas *qweb* usado por el controlador de rutas.
- *__manifest__.py*: es el manifiesto del módulo. Incluye información como el título, descripción, así como ficheros a cargar. En el ejemplo se debe descomentar la línea que contiene la lista de control de acceso en el fichero *security/ir.model.access.csv*

6.3 Actualización de módulos

Como en cualquier proceso de desarrollo, es necesario que los cambios efectuados en el código aparezcan en reflejados en el servidor.

Odoo tiene un forma de funcionar *data-driven* (dirigido por datos) y cuando instalamos un módulo, las vistas, datos, etc. (los *.xml*) se almacenan en la base de datos. La actualización de la base de datos con datos, vistas, etc. sólo se hace **al instalar o actualizar el módulo**. Así que aunque cambiemos una vista en un fichero XML, no se verá el cambio si no actualizamos el módulo *Odoo* (ni siquiera reiniciando el servicio *Odoo*).

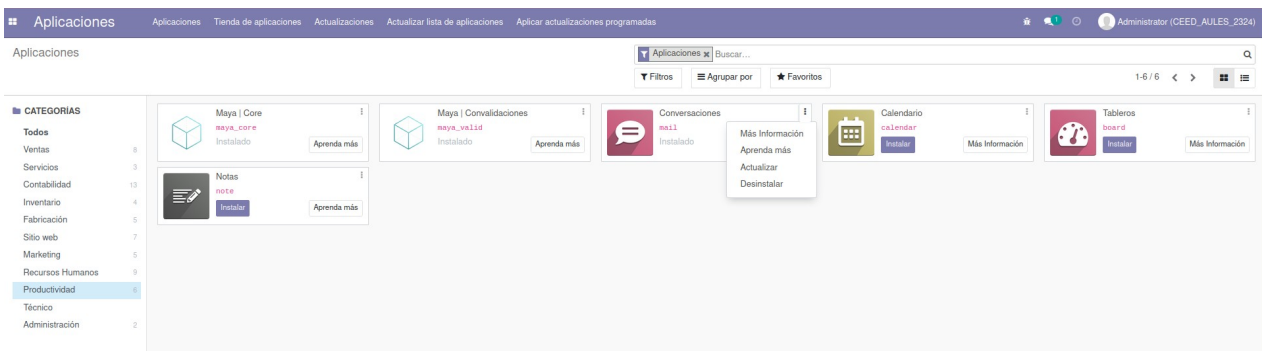


En caso de estar utilizando Docker, es posible reiniciar el contenedor con el comando: `docker compose restart web`



En caso de estar utilizando Odoodock, es posible reinicar el contenedor con el comando: `docker compose down`, aunque generalmente con reiniciar la instancia de depuración ya hay mas que suficiente.

Los ficheros *Python* de un módulo son cargados de nuevo cada vez que se inicia el servicio *Odoo*. Por lo que si cambiamos algo en ellos, tenemos dos opciones para observar los cambios: recargar el módulo o reiniciar el servicio *Odoo* (si necesidad de actualizar el módulo).



Si estamos trabajando en modo desarrollo (opción `--dev=all` al lanzar *Odoo*), el servidor *Odoo* leerá cada vez las vistas, datos y código *Python* directamente de los ficheros. Por lo tanto, podremos observar cambios de vistas, datos o código sin reiniciar el servicio ni actualizar el módulo.



Esta práctica es muy cómoda para desarrollar, pero no se usa en entornos de producción por motivos tanto de seguridad como de rendimiento.



Aunque no ocurre muchas veces, el modo desarrollo no siempre funciona como se le espera, siendo necesario en ese caso actualizar el módulo (o reiniciar el servicio). Estos fallos suelen aparecer casi siempre cuando el código afecta de alguna manera a la estructura de la base de datos.

7 FICHERO `__manifest__.py`

La función del fichero `__manifest__.py` es doble. Por una parte sirve para indicar que un módulo (paquete) de *Python* es un módulo de *odoo* y por otra para definir los metadatos del módulo. Su contenido es un diccionario *Python* donde cada una de las claves especifica un metadato.

Las claves disponibles son:

- `name (str)`. Nombre comercial del módulo (es un campo obligatorio)
- `version (str)`. Versión del módulo. Tiene que seguir las normas del versionado semántico.
- `summary (str)`. Resumen corto de la funcionalidad del módulo.

- `description (str)`. Descripción extendida del módulo.
- `author (str)`: autor del módulo.
- `website (str)`: url con información ya sea sobre el módulo o sobre el autor.
- `license (str)`. Tipo de licencia de la distribución. Las opciones disponibles son: *GPL-2, GPL-2 or any later version, GPL-3, GPL-3 or any later version, AGPL-3, LGPL-3, Other OSI approved licence, OEEL-1 (Odoo Enterprise Edition License v1.0), OPL-1 (Odoo Proprietary License v1.0), Other proprietary*. Por defecto: *LGPL-3*
- `category (str, default: Uncategorized)`. Categoría donde se va a incluir el proyecto. En principio se permite cualquier texto, de manera que si la categoría no existe se creará en ese momento. Aún así, existen una serie de categorías de serie que pueden utilizarse.
- `depends (list(str))`. Una lista [], de otros módulos que son necesarios para el funcionamiento del módulo. Estos módulos deben ser cargados antes de la carga del módulo actual.



Cuando se instala un módulo, todas sus dependencias se instalan antes que él. Del mismo modo, las dependencias se cargan antes de que se cargue un módulo.



El módulo *base* siempre está instalado, pero es necesario incluirlo en la lista de dependencias para asegurar que nuestro módulo es actualizado cuando el módulo *base* recibe una actualización..

- `external_dependencies (dict(key=list(str)))`. Un diccionario que contiene dependencias externas, ya sean ficheros de *python* o *ejecutables*. Su estructura consiste en un dos claves *python* y *bin*, donde cada una de ellas almacena una lista con las dependencias. Por ejemplo:

```
'external_dependencies': {
    'python': [ 'toolz' ],
    'bin': [ 'pdftk' ],
}
```



Al contrario que con las dependencias de módulos (*depends*), Odoo no instala las dependencias externas, únicamente comprueba de su existencia en el sistema.

- `data (list(str))`. Lista de ficheros que serán siempre instalados o actualizados junto con el módulo, básicamente ficheros *xml* y *csv*. Hay que indicar la ruta del fichero desde el directorio raíz del módulo.

```
'data': {
    'security/security_groups.xml',
    'reports/custom_footer.xml',
    'data/departaments.xml',
}
```

- `demo (list(str))`: lista de ficheros para instalar sólo en modo demostración.
- `auto_install (bool)`. Indica si el módulo debe ser instalado o actualizado, cuando es dependencia de otro. Puede tener implicaciones en el comportamiento del sistema ya que podría instalar módulos sin el conocimiento o consentimiento explícito del usuario. Por defecto: False.
- `application (bool)`. Indica si el módulo es considerado una aplicación en si misma o un módulo de apoyo a otros módulos aplicación `is just a technical module`. Por defecto: False
- `assets (dict)`. Se usa para especificar recursos estáticos como CSS, imágenes, ficheros javascript, etc. Se agrupan en los llamados *bundles*. Por ejemplo, `web.assets_common` es el bundle que agrupa los elementos comunes al cliente web, la web externa o el TPV, mientras que `web.assets_frontend` contiene los específicos de la web externa (blog, ecommerce...)

```
'assets': {
    'web.assets_common': [
        'mimodulo/static/css/mi_estilo.css'
    ],
}
```



Es posible encontrar más información sobre *assets* y *bundles* en la [documentación oficial de Odoo](#).

- `installable (bool)`. Indica si un usuario debe ser capaz de instalar el módulo desde la interfaz web. Por defecto: True.
- `maintainer (str)`. Persona o entidad encargada del mantenimiento. Por defecto se asume que es el autor.
- `{pre_init, post_init, uninstall}_hook (str)`. Los *hooks* son funciones que son llamadas en diferentes procesos de la vida del módulo, en concreto antes de instalar, después de instalar y después de la desinstalación. Las funciones tienen que estar definidas en el fichero `__init__.py` del módulo.

```
'pre_init_hook': '_init_hook'
```

8 EJEMPLO COMPLETO: LISTA DE TAREAS

Aunque en el tema siguiente entraremos en profundidad en el desarrollo, vamos a poner en marcha un primer módulo funcional. Para ello vamos a utilizar un ejemplo comentado donde crearemos una sencilla *Lista de tareas*.

Este módulo tendrá una estructura similar a esta:

Tarea	Prioridad	Urgente	Realizada
<input type="checkbox"/> Hacer la compra	11	<input checked="" type="checkbox"/>	<input type="checkbox"/>
<input type="checkbox"/> Arreglar mi ordenador.	5	<input type="checkbox"/>	<input type="checkbox"/>

Y nos permitirá crear tareas con una prioridad asociada. También podremos marcar si la tarea esta realizada o no. El campo *urgente* será un campo calculado (es decir, no se podrá editar a mano) que estará marcado si la prioridad es mayor que 10.

A continuación, desglosamos el código comentado de esta aplicación:

Fichero `__manifest__.py`:

```
# -*- coding: utf-8 -*-
{
    'name': "Lista de tareas",

    'summary': """
Sencilla Lista de tareas""",

    'description': """
Sencilla lista de tareas utilizadas para crear un nuevo módulo con un
nuevo modelo de datos
""",

    'author': "Pepo Sánchez",
    'website': "https://apuntesfpinformatica.es",
    #Indicamos que es una aplicación
    'application': True,

    # Vamos a utilizar la categoría Productivity
    'category': 'Productivity',
```

```
'version': '0.1',

# Indicamos lista de módulos necesarios para que este funcione
correctamente
# En este ejemplo solo depende del módulo "base"
'depends': ['base'],

# Esto siempre se carga
'data': [
#Este primero indica la politica de acceso del módulo
'security/ir.model.access.csv',
#Cargamos las vistas y las plantillas
'views/views.xml',
]
}
```

Fichero **models.py**:

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

from odoo import models, fields, api

#Definimos el modelo de datos
class lista_tareas(models.Model):
    #Nombre y descripcion del modelo de datos
    _name = 'lista_tareas.lista_tareas'
    _description = 'lista_tareas.lista_tareas'

    #Elementos de cada fila del modelo de datos
    #Los tipos de datos a usar en el ORM son
    # https://www.odoo.com/documentation/14.0/developer/reference/addons/orm.html#fields

    tarea = fields.Char()
    prioridad = fields.Integer()
    urgente = fields.Boolean(compute="_value_urgente", store=True)
    realizada = fields.Boolean()

    #Este computo depende de la variable prioridad
    @api.depends('prioridad')
    #Funcion para calcular el valor de urgente
    def _value_urgente(self):
```

```
#Para cada registro
for record in self:
    #Si la prioridad es mayor que 10, se considera urgente, en otro caso no lo es
    if record.prioridad>10:
        record.urgente = True
    else:
        record.urgente = False
```

Fichero views.xml:

```
<odoo>
  <data>
    <!-- explicit list view definition -->
    <!-- Definimos como es la vista explicita de la litas-->
    <record model="ir.ui.view" id="lista_tareas.list">
      <field name="name">lista_tareas list</field>
      <field name="model">lista_tareas.lista_tareas</field>
      <field name="arch" type="xml">
        <tree>
          <field name="tarea"/>
          <field name="prioridad"/>
          <field name="urgente"/>
          <field name="realizada"/>
        </tree>
      </field>
    </record>
    <!-- actions opening views on models -->
    <!-- Acciones al abrir las vistas en los modelos
    https://www.odoo.com/documentation/14.0/developer/reference/addons/
    actions.html
    -->
    <record model="ir.actions.act_window" id="lista_tareas.action_window">
      <field name="name">Listado de tareas pendientes</field>
      <field name="res_model">lista_tareas.lista_tareas</field>
      <field name="view_mode">tree,form</field>
    </record>

    <!-- Top menu item -->
```



```
<menuitem name="Listado de tareas" id="lista_tareas.menu_root"/>

<!-- menu categories -->
<menuitem name="Opciones Lista Tareas" id="lista_tareas.menu_1"
parent="lista_tareas.menu_root"/>

<!-- actions -->
<menuitem name="Mostrar lista" id="lista_tareas.menu_1_list"
parent="lista_tareas.menu_1"
    action="lista_tareas.action_window"/>

</data>
</odoo>
```

9 BIBLIOGRAFIA

1. Sistemas de Gestió Empresarial IOC
2. Wikipedia
3. Documentación de Odoo

10 AUTORES

A continuación ofrecemos en orden alfabético (por apellido) el listado de autores que han hecho aportaciones a este documento.

- Jose Castillo Aliaga
- Sergi García Barea
- Alfredo Oltra

Gran parte del contenido ha sido obtenido del material con licencia CC BY SA disponible en LearnXinYminutes.