

## **UNIDAD 5: DESARROLLO DE COMPONENTES**

**Módulo profesional:**  
**Sistemas de gestión empresarial**

# Índice

RESUMEN INTRODUCTORIO.....	3
INTRODUCCIÓN.....	3
CASO INTRODUCTORIO .....	4
1. DESARROLLO DE COMPONENTES PARA UN SISTEMA ERP-CRM.....	5
1.1 Técnicas y estándares .....	6
1.2 Especificaciones funcionales para el desarrollo de componentes .....	8
2. DESARROLLO DE MÓDULOS .....	11
2.1. Entornos y herramientas de desarrollo en sistemas ERP y CRM .....	12
2.2. Lenguaje proporcionado por los sistemas ERP-CRM .....	15
2.2.1. Características y sintaxis del lenguaje .....	16
2.2.2. Declaración de datos .....	16
2.2.3. Operadores .....	17
2.2.4. Estructuras de programación y sentencias del lenguaje .....	18
2.3. Formularios e informes en sistemas ERP-CRM. ....	20
2.3.1. Llamadas a funciones, librerías de funciones.....	21
2.3.2. Elementos principales .....	22
2.3.3. Inserción, modificación y eliminación de datos .....	25
2.3.4. Acciones y menús .....	27
2.3.5. Seguridad .....	28
2.4. Extracciones de informaciones contenidas en sistemas ERP-CRM....	32
2.4.1. Herramientas para la creación de formularios e informes .....	33
2.4.2. Diseño de informes .....	34
2.5. Operaciones de consulta. Herramientas.....	35
2.5.1. Técnicas de optimización de consultas y acceso a grandes volúmenes de información.....	39
2.5.2. Generación de programas de extracción de datos entre sistemas (batch inputs) .....	41
2.6. Depuración de un programa .....	42
2.6.1. Manejo de errores .....	44
RESUMEN FINAL .....	46

## RESUMEN INTRODUCTORIO

En esta unidad, finalizamos el estudio en la implantación de un ERP-CRM profundizando en los conceptos básicos y herramientas sobre desarrollos de componentes y módulos.

Comenzaremos dando las pautas sobre cuando una empresa se plantea la realización de desarrollos a medida y cómo puede planificarlas.

En segundo lugar, entraremos de lleno en la realización de un nuevo módulo dentro de Odoo, cuáles son las herramientas necesarias y los ficheros implicados. Un módulo incorpora modelos que y controladores que podrán ser después utilizados en diferentes vistas como formularios, informes o gráficos.

Por último, introducimos los conceptos sobre depuración y cuáles son las metodologías y herramientas que disponemos para la gestión de los errores.

## INTRODUCCIÓN

Uno de los motivos por los que una empresa se plantee realizar desarrollos a medida es la adaptación de los procesos dentro de la misma. Un desarrollo a medida puede tener diferentes alcances, desde el desarrollo desde cero y completo a la adaptación de un software estándar.

Dentro de Odoo tenemos la opción de estudiar y realizar pruebas sobre ese desarrollo de nuevas utilidades y características a través de la creación de nuevos módulos. Dentro de un módulo de Odoo podremos definir modelos, datos, vistas e informes y de esta forma poder adaptar las necesidades de la empresa al software.

Una vez que nos encontramos desarrollando aplicaciones, es importante conocer las herramientas que nos permiten realizar debug para poder reaccionar sobre los errores propios del desarrollo.

## CASO INTRODUCTORIO

Como desarrollador de aplicaciones perteneces al departamento de informática de una cadena de franquicias dentro del sector de restauración y hostelería con 15 restaurantes temáticos a lo largo del país.

Se ha comenzado realizando una instalación del sistema ERP de Odoo en su versión community en un servidor para poder realizar pruebas, se han configurado varios usuarios, entre ellos uno de administrador y desarrollador.

Junto a tu jefe de departamento necesitáis realizar una evaluación de un desarrollo a medida, ya que el proceso de procesado de productos en cocina y distribución a las diferentes franquicias no viene como un estándar.

Al final de esta unidad tendremos los conocimientos para poder realizar un nuevo módulo a medida, así como sus vistas pertinentes.

## 1. DESARROLLO DE COMPONENTES PARA UN SISTEMA ERP-CRM

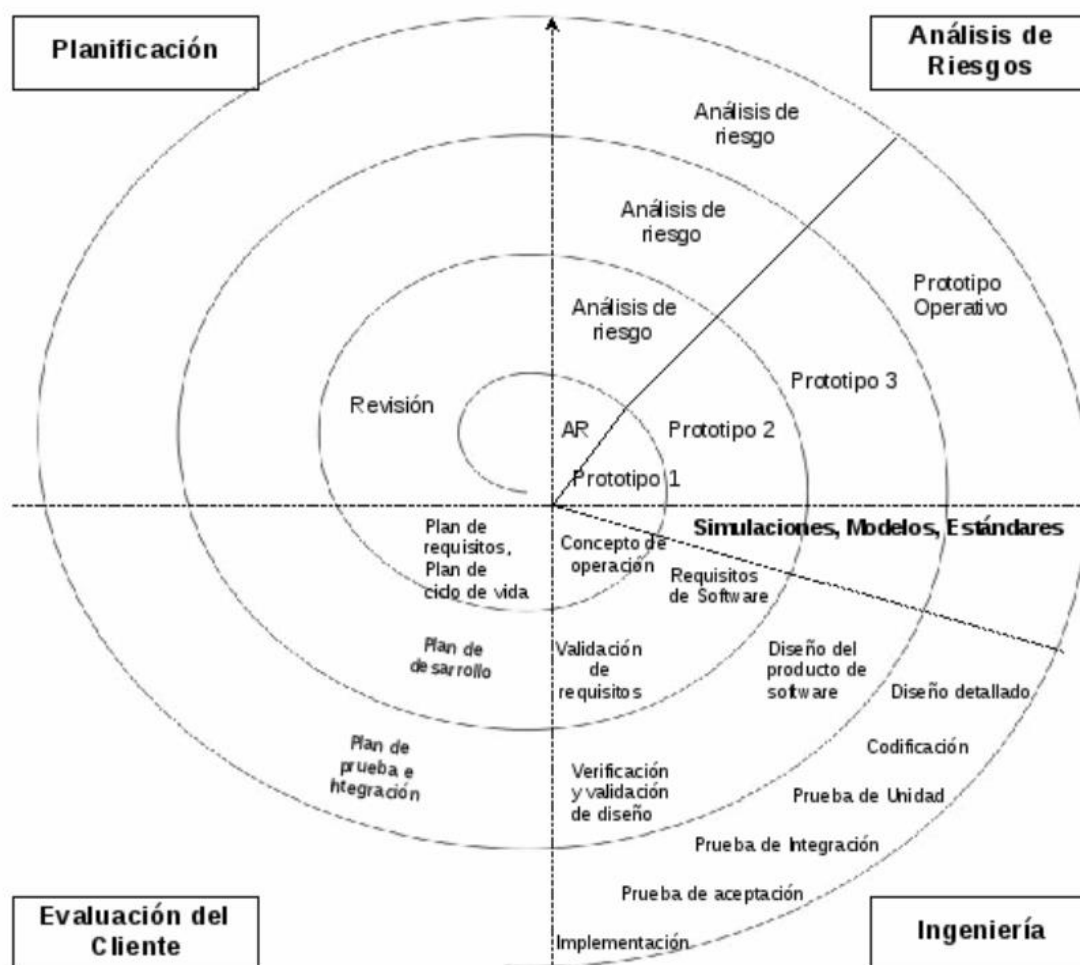
*El primer paso, es analizar las necesidades que tiene una empresa, departamento a departamento y proceso a proceso, conociendo también que posibilidades nos da un sistema abierto como es Odoo ERP.*

*Tú y tu jefe para el desarrollo de ese módulo os planteáis varias preguntas ¿Cuánto tiempo llevará realizar el desarrollo? ¿Qué recursos necesitamos tener en cuenta? ¿Qué personal no técnico debe estar implicado?*

*A partir de estas dudas es donde se plantea realizar una planificación del desarrollo.*

Tal y como hemos desarrollado e introducido a lo largo del curso en anteriores unidades didácticas, las empresas tienen necesidades diversas que hacen que la implantación de un sistema de gestión empresarial ERP-CRM necesite en la gran mayoría de ocasiones de adaptaciones para poder dar respuesta a los diferentes procesos y operaciones que se realizan dentro del negocio.

Además, estas necesidades son vivas, ya que evolucionan de la misma forma que lo hace una empresa. Al comienzo de la evaluación de un software de gestión empresarial se resumen todas las necesidades a través de la captura de requisitos de un determinado sistema. Durante las pruebas e implantación aparecen nuevos requisitos y modificaciones posibles. También a lo largo del uso y evolución de la empresa pueden aparecer nuevas funcionalidades y requisitos a desarrollar.



Modelo en espiral desarrollado por Boehm

Fuente: <https://muhas.org/modelos-de-proceso-del-software.html>

Como vemos en la anterior imagen, un posible modelo de planificación de proyectos consiste en un modelo en espiral donde las fases se plantean una y otra vez al pasar por el mismo estado, pero en diferentes momentos de la vida de una empresa.

## 1.1 Técnicas y estándares

Podemos entender el software empresarial desde tres puntos de vista:

- El software standard es el que se ha pensado y desarrollado para poderse implantar en cualquier tipo de empresa, sector o actividad, intentando que se cubran los procesos de gestión más habituales dentro de las empresas. Dentro de estos softwares generalistas nos encontramos las llamadas verticalizaciones que consiste en una especialización dentro de un determinado sector. Por ejemplo, una verticalización de software de gestión dentro de las empresas del

sector de la moda puede llevar desarrollado de serie muchos aspectos sobre el tallaje o la venta.

- El siguiente paso dentro de los diferentes tipos de software de gestión tenemos el software híbrido. Durante la captura de necesidades dentro de una empresa se recogen esas necesidades que se convierten en adaptaciones a partir de los diferentes módulos que tiene el software, donde esas adaptaciones a su vez pueden ser de dos niveles, modificaciones sobre ventanas, vistas e informes existentes, o generación de nuevos elementos que se puedan añadir a nuestro sistema.
- Por último, el software a medida es la tercera opción que nos encontramos de tipología de software de gestión. En este caso el planteamiento desde el principio es el desarrollo del software desde cero y a medida para los procesos de negocio de la empresa en concreto. Este caso, debe ser tomado como recurso después de haber realizado un análisis minucioso de las opciones que ofrece el mercado, ya que el desarrollo a medida normalmente es costoso en tiempo y en recursos.

Las técnicas y estándares que se utilizan en la adaptación y desarrollo a medida de software de gestión empresarial son las mismas que en el desarrollo de software. Se utilizan lenguajes de programación libre o código abierto, como es el caso de Odoo que utiliza los siguientes lenguajes de programación:

- Lenguaje Python para el desarrollo de la lógica computacional.
- Lenguajes HTML, JS y CSS para el desarrollo visual y de interacción con el usuario.
- PostgreSQL para la gestión de la base de datos.

Una de las metodologías que más de moda nos encontramos actualmente y más utilizadas para la planificación del desarrollo software son las metodologías ágiles, SCRUM y KANBAN entre muchas de ellas.



#### **ENLACE DE INTERÉS**

Es esencial visitar el manifiesto ágil para comenzar a comprender las metodologías de gestión de proyectos como SCRUM:

<http://agilemanifesto.org/iso/es/manifesto.html>



### COMPRUEBA LO QUE SABES

Acabamos de estudiar los diferentes lenguajes que tenemos dentro del software de gestión Odoo.

¿Serías capaz de diferenciar los diferentes lenguajes Python, XML, HTML, CSS, JS? Su uso dentro de Odoo. Sus diferencias como lenguajes.

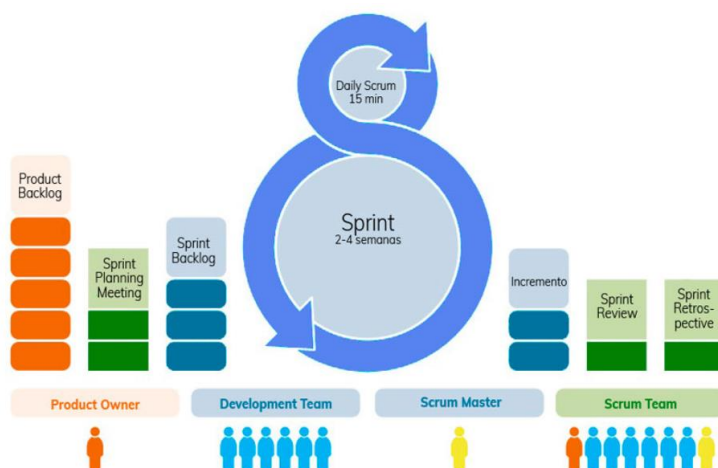
Coméntalo en el foro.

## 1.2 Especificaciones funcionales para el desarrollo de componentes

El esfuerzo que supone realizar un documento de especificaciones detallado junto con los requisitos es muy grande. Es importante tener dentro de los hitos este objetivo, pero puede que ni el cliente ni nosotros si somos los generadores del proyecto, sepamos de todas las necesidades a priori.

En muchas ocasiones, los usuarios necesitan ver físicamente el producto para poder plantear necesidades o modificaciones, y estas necesidades puede que en muchos casos aparezcan durante el uso del software, y más en el caso del software de gestión empresarial que tiene tantos pequeños detalles que repercuten enormemente en el funcionamiento final.

Hemos introducido las metodologías ágiles y en concreto SCRUM como herramienta para la planificación de nuestro proyecto.



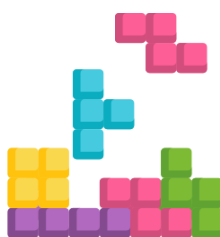
Metodología SCRUM

Fuente: <https://jorgesaiz.com/blog/proyecto-scrum-una-explicacion-sencilla/>



Como vemos en la imagen y de una forma sencilla, en SCRUM los requisitos se resuelven con el elemento denominado Product BackLog. Es una lista viva de los requisitos funcionales y no funcionales que dan valor a un cliente y al producto y que además se encuentran priorizados.

Sin entrar en el detalle del funcionamiento de SCRUM, esta lista es viva ya que a lo largo del desarrollo del proyecto van actualizándose o modificándose a lo largo de lo que se denominan Sprints. Un Sprint como vemos tiene la capacidad de planificar parte del proyecto a lo largo de un período de semanas y que nos permite hacer visible parte del proyecto. Esa visibilidad a su vez nos permite un feedback sobre parte del proyecto desde el punto de vista del cliente y los usuarios y por lo tanto poder modificar esos requisitos iniciales y las planificaciones futuras en forma de nuevos sprints.



## EJEMPLO PRÁCTICO

En la empresa donde estamos trabajando como desarrolladores de software, vamos a realizar un nuevo módulo sobre Menús y Platos. Para ello necesitamos obtener los requisitos del software para construir nuestro producto backlog.

- 1) En el equipo de elaboración de los requisitos se deben reunir, tanto el cliente (Product Owner) como el responsable de desarrollo (Scrum Master). En nuestro caso, el Product Owner será el director de restaurantes y franquicias, y como Scrum Master el jefe de departamento técnico.
- 2) En segundo lugar, recogeremos todas las tareas (historias de usuario en argot SCRUM), con un título suficientemente claro y específico, y un detalle del mismo. Un ejemplo de un requisito tendría las siguientes partes:
  - a. **Historia:** Crear un plato.
  - b. **Como:** Cocinero.
  - c. **Quiero:** Introducir todos los ingrediente de un plato con un nombre y título de plato.

*Historia: Agregar comentarios*

*Como: Lector del Blog*

*Quiero: adicionar comentarios a las entradas y recibir alertas cuando otros hagan comentarios*

*Para: mantenerme en contacto con los demás usuarios del blog*

**3**

*Historia: Responder a comentarios*

*Como: Lector del Blog*

*Quiero: adicionar comentarios a las entradas y responder a comentarios de otros lectores*

*Para: mantenerme en contacto con los demás usuarios del blog*

**3**

### Ejemplo de historia de usuario

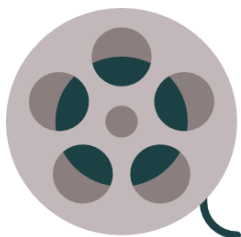
Fuente: <https://www.blmovil.com/scrum-historias-de-usuario-ii/>

- 3) Priorizamos la lista de requisitos (historias de usuario), bien de una forma secuencial, bien agrupándola o bien mediante una estimación de tiempos.

Área de requisitos	Requisitos	Origen	Valor	Estimación inicial	Factor Ajuste	Estimación ajustada	Iteración: Pendiente: 225 170 114 57				
Área X	Requisito A	Marketing	2000	15		15	15	0			
Área Z	Requisito B	Producción	1750	20		20	20	0			
Área Y	Requisito C	Ventas	1500	20		20	20	0			
	Iteración 1		5250	55		55	55	0	0	0	0
Área Z	Requisito C	Producción	1250	15	0,2	18	18	18	0		
Área X	Requisito D	Producción	1250	20		20	20	20	0		
Área Z	Requisito E	Marketing	1000	15	0,2	18	18	18	0		
	Iteración 2		3500	50		56	56	56	0	0	0
	Primera entrega		8750	105		111	111	56	0	0	0
Área X	Requisito F	Marketing	1250	20	0,2	24	24	24	24	0	
Área Y	Requisito G	Marketing	750	15		15	15	15	15	0	
Área Y	Requisito H	Ventas	750	15	0,2	18	18	18	18	0	
	Iteración 3		2750	50		57	57	57	57	0	0
Área Z	Requisito I	Producción	700	15	0,2	18	18	18	18	18	
Área Y	Requisito J	Marketing	500	10	0,5	15	15	15	15	15	
Área Y	Requisito K	Ventas	500	20	0,2	24	24	24	24	24	
	Iteración 4		1700	45		57	57	57	57	57	0
	Segunda entrega		4450	95		114	114	114	114	57	0

### Ejemplo de priorización

Fuente: <https://proyectosagiles.org/lista-requisitos-priorizada-product-backlog/>



### VIDEO DE INTERÉS

A continuación se muestra un vídeo muy explicativo de las partes más importantes de la metodología SCRUM:

<https://www.youtube.com/watch?v=P25JP0u6UKw>

## 2. DESARROLLO DE MÓDULOS

*Para la realización del desarrollo, es necesario conocer todas las posibilidades que proporciona el software de gestión empresarial, las herramientas necesarias, la tecnología y cómo realizarlo.*

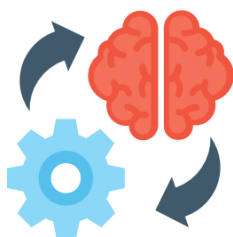
*Tu jefe y tú os planteáis varias preguntas ¿Cómo poder ampliar las características del software? ¿Cómo ampliar las capacidades de información de los módulos? ¿Cómo realizar nuevas vistas?*

*A partir de estas dudas se comienza un trabajo de investigación, desarrollo y prueba de un nuevo módulo con Odoo.*

Una solución híbrida tal y como acabamos de estudiar es una opción fantástica y muy válida para poder tener lo mejor de las soluciones a medida y de las soluciones estándar. Cuando una empresa o un negocio se plantea implantar un ERP-CRM realiza un proceso de estado del arte de este tipo de software, al mismo tiempo que realiza un estudio de las necesidades que transforma en unas especificaciones para el proyecto de implantación.

Aun cuando este proceso se realice muy minuciosamente y realizando pertinentes pruebas sobre el software, se producen durante la implantación y sobre todo durante el uso del software de gestión momentos en los que se observa necesidades nuevas no tenidas en cuenta. Estas nuevas necesidades que no proporciona el software se pueden resolver de dos formas con Odoo, mediante modificaciones de los formularios, informes o vistas tal y como hemos visto anteriormente, o mediante el desarrollo de nuevos módulos.

Un módulo dentro de Odoo nos permite modificar y personalizar el comportamiento de la instalación estándar de Odoo, añadiendo nuevas funcionalidades o alterando funcionalidades existentes que ya teníamos con otros módulos.

**RECUERDA**

En la UD4 en el punto 2 vimos cómo se realizaba el proceso de parametrización y adaptación dentro de Odoo, tanto con las tablas y vistas añadiendo nuevos elementos como con los informes modificándolos.

En la UD2 en el punto 2.5.7 introdujimos el proceso de implantación y dentro de este las medidas a tener en cuenta, medidas como la parametrización o la adaptación.

## 2.1. Entornos y herramientas de desarrollo en sistemas ERP y CRM

Para el desarrollo de un nuevo modelo dentro de Odoo necesitaremos las siguientes herramientas y entornos, algunos de ellos ya los hemos trabajado en unidades anteriores:

- Odoo community, el framework nos proporciona lo esencial para poder comenzar a realizar las pruebas y desarrollos ya que tal y como hemos visto es un software de gestión empresarial de código abierto.
- Un IDE de desarrollo que nos permita editar el código, así como nos proporcione utilidades para realizar esa tarea de forma cómoda. En nuestro caso usaremos Visual Code con diversos widgets/plugins instalados para realizar dicho desarrollos. Hemos de tener en cuenta que para realizar las modificaciones y crear nuevos módulos nos encontramos con múltiples lenguajes, por lo que Visual Code será una herramienta perfecta para llevar a cabo esos desarrollos.
- Compiladores y entornos de desarrollo relacionados con los lenguajes a programar. Algunos lenguajes, en nuestro caso Python, necesitan de instalaciones añadidas para poder realizar las pruebas, depuraciones y compilaciones del código desarrollado.

Tal y como comentamos, Visual Studio Code es un entorno de programación modular, ligero y gratuito que nos permite realizar programación con diferentes lenguajes de programación.

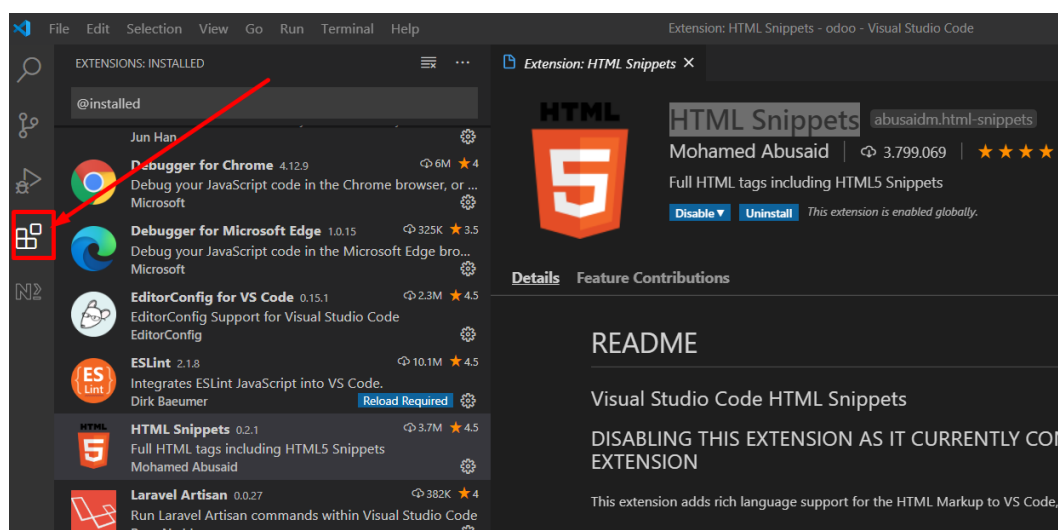


## ENLACE DE INTERÉS

Enlace a la página del fabricante de Visual Studio Code y su descarga.

<https://code.visualstudio.com/>

En nuestro caso, dentro del desarrollo de módulos de Odoo necesitaremos instalar diferentes plugins que nos faciliten las tareas como desarrolladores. Para ello una vez abierto Visual Code nos dirigiremos a la zona de gestión de extensiones tal y como vemos en la imagen:



Extensiones Visual Code

Fuente: imagen propia

Dentro de las extensiones recomendadas que debemos instalar son:

- Extensiones para el reconocimiento y uso de los lenguajes HTML, CSS y JS. Entre ellas nos encontramos con HTML Snippets o ES Lint.
- Extensiones específicas de Python como Python extensión.



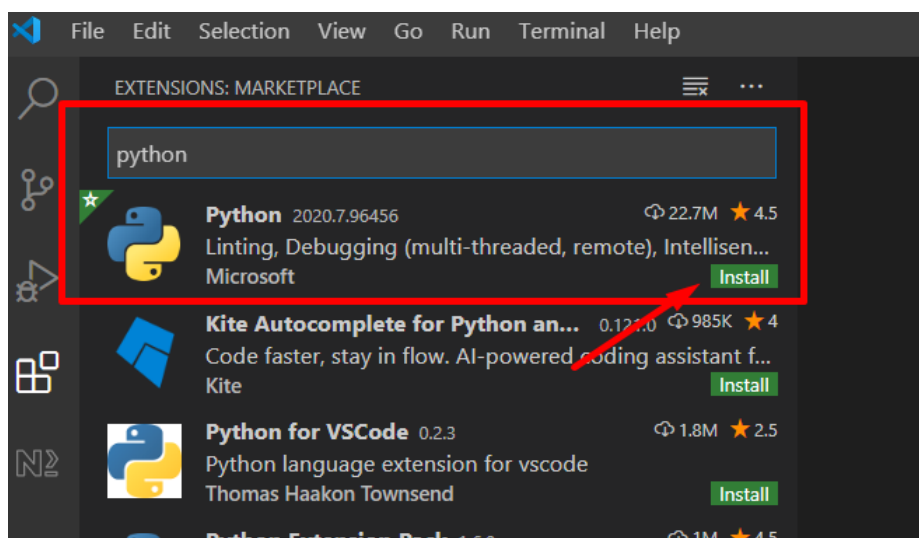
## ENLACE DE INTERÉS

Desde el propio entorno de desarrollo, Visual Code Studio podemos realizar la instalación de los diferentes plugins de desarrollo:

- Python, <https://marketplace.visualstudio.com/items?itemName=ms-python.python>
- HTML, <https://marketplace.visualstudio.com/items?itemName=abusaidm.html-snippets>
- JS, <https://marketplace.visualstudio.com/items?itemName=dbaumer.vscod-e-eslint>

Un ejemplo de instalación de los plugins lo podemos tener con el plugin de Python:

1. Haremos clic sobre el icono de extensiones de Visual Code Studio.
2. En la búsqueda de extensiones escribiremos Python e instalaremos la extensión denominada Python.



Instalación de Python

Fuente: imagen propia

3. Es recomendable cambiar la versión de Python que estamos usando en nuestro configurador de extensión. Teclear (CTRL+Shift+P), aparecerá un menú sobre el que escribiremos, *Python: Select Interpreter*. Seleccionaremos el intérprete de Python instalado.

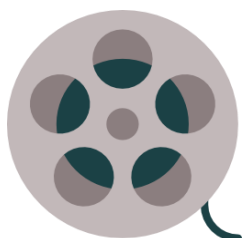


### COMPRUEBA LO QUE SABES

Acabamos de estudiar el entorno de desarrollo Visual Code Studio y los plugins necesarios para un buen desarrollo.

¿Serías capaz de encontrar otro entorno de desarrollo que nos pudiera servir para su uso con Odoo? Marca ventajas y desventajas con Visual Code.

Coméntalo en el foro.



### VIDEO DE INTERÉS

En el siguiente vídeo tenemos cómo utilizar Visual Studio con Python.

<https://www.youtube.com/watch?v=9Hh6fqieReE>

## 2.2. Lenguaje proporcionado por los sistemas ERP-CRM

Son muchos los lenguajes de programación que existen y utilizan los diferentes software de gestión empresarial. En el caso de Odoo los diferentes lenguajes de programación que se usan son:

- **HTML y CSS**, son lenguajes de marcas usados en el desarrollo de aplicaciones web. En el caso de Odoo estos 3 lenguajes se utilizan para el desarrollo de la parte frontal o de vistas de la aplicación. No son lenguajes de programación estrictamente.
- **JavaScript**, es un lenguaje de programación interpretado por el navegador que también nos sirve para definir la parte frontal y de vistas de nuestra aplicación.
- **Python**, es un lenguaje de programación de alto nivel, interpretado y orientado a objetos. En el caso de Odoo, se utiliza el lenguaje para todos los procesos de lógica.

### 2.2.1. Características y sintaxis del lenguaje

Entre sus principales características destacan:

- **Portabilidad:** es posible ejecutarlo en diversas plataformas como Unix, Windows, OS/2 (ver glosario), etc.
- **Facilidad de uso:** posee una sintaxis muy sencilla por lo que es muy fácil su aprendizaje.
- **Flexibilidad:** dispone de interfaces a llamadas al sistema y bibliotecas.
- **Interactividad.**
- **Gratuidad** y libre distribución.
- **Orientación a objetos:** como hemos comentado es un lenguaje orientado a objetos y por tanto dispone de herencia múltiple, polimorfismo, ligadura dinámica, así como todas las características de los lenguajes OO.
- **Lenguaje de alto nivel:** y como tal, facilita el no preocuparnos de detalles de bajo nivel tales como manejar la memoria empleada por el programa.
- **Ampliable:** dispone de multitud de librerías, tipos de datos y funciones incorporadas que facilitan la realización de tareas habituales sin necesidad de programarlas desde el inicio.



#### ENLACE DE INTERÉS

En el siguiente enlace encontramos un manual de Python para principiantes que hace un recorrido por toda la sintaxis y usos de este lenguaje.

<https://uniwebsidad.com/libros/python?from=librosweb>

### 2.2.2. Declaración de datos

Los tipos de datos en Python se dividen en mutables, si su contenido se puede modificar, e inmutables, si su contenido no puede alterarse.

- **Números:** todos son tipos de datos inmutables.
  - Enteros (int)
  - Reales (float)
  - Complejos (complex)
- **Constantes lógicas** (bool): todas son del tipo inmutables.



- **Secuencias**

- Cadenas (string): pueden estar delimitadas por comillas simples o dobles ('Hola', "Hola")
- Tuplas (tuple): se delimitan por paréntesis y sus elementos se indican separados por comas (a,b,c).
- Listas (list): se delimitan por corchetes y sus elementos se indican separados por comas[1, 2, 3].
- Las cadenas y tuplas son datos del tipo inmutables y las listas mutables.

- **Conjuntos:** colecciones no ordenadas de elementos (inmutables) que no se pueden encontrar duplicados.

- Para que el conjunto sea mutable utilizamos la sentencia: set(secuencia) la cual crea un conjunto con los elementos de la secuencia especificada descartando las duplicaciones.
- Para que el conjunto sea inmutable utilizamos la sentencia: frozenset(secuencia) la cual crea un conjunto con los elementos de la secuencia especificada descartando las duplicaciones.

- **Otros tipos de datos** La constante None se utiliza para especificar un dato nulo.

### 2.2.3. Operadores

Sin entrar en profundizar en el uso de Python, las operaciones y estructuras dentro de Python son:

- **Comparadores lógicos:** en la siguiente imagen nos encontramos con un resumen de los diferentes comparadores.

<i>Operador</i>	<i>Significado</i>
<	Menor estricto que
<=	Menor o igual que
>	Mayor estricto que
>=	Mayor o igual que
==	Igual que
!=	Distinto que
is	Idéntico a
is not	No idéntico a

Comparadores en Python

Fuente: Coremsa

- **Operadores lógicos:** en Python existen tres operadores lógicos o booleanos: not, and, or.

<i>Declaración</i>	<i>Evalúa a</i>
bool(expr)	True si expr es verdadera, False en caso contrario
not expr	True si expr es falsa, False en caso contrario
expr1 or expr2	False si expr1 y expr2 son falsos, True en caso contrario
expr1 and expr2	True si expr1 y expr2 son verdaderos, False en caso contrario

Operadores lógicos en Python

Fuente: Coremsa

- **Operadores de asignación:** permiten la asignación de elementos.

<i>Operador</i>	<i>Resultado</i>
a=b	Asigna el dato b a la etiqueta a
a+=b	Lo mismo que a=a+b
a-=b	Lo mismo que a=a-b
a*=b	Lo mismo que a=a*b
a/=b	Lo mismo que a=a/b
a%=b	Lo mismo que a=a %b
a**=b	Lo mismo que a=a**b

Operadores de asignación en Python

Fuente: Coremsa

#### **2.2.4. Estructuras de programación y sentencias del lenguaje**

Dentro de las posibilidades que nos encontramos dentro Python vamos a tratar de forma resumida tres de los aspectos más importantes.

- **Control de flujo,** son estructuras de programación que nos encontramos en todos los lenguajes y que nos permiten definir y controlar el flujo de la ejecución de las sentencias. En la siguiente imagen tenemos un resumen de las estructuras más importantes dentro de Python.

<i>Declaración</i>	<i>Significado</i>
<b>if</b> condicion: consecuencias <b>elif</b> condicion: consecuencias <b>else</b> : alternativas	Condicionales simple ( <b>if</b> ), doble ( <b>if/else</b> ) y múltiple ( <b>if/elif/else</b> )
<b>while</b> condicion: acciones	Bucle mientras
<b>for</b> elt <b>in</b> secuencia: acciones	Bucles para y desde
<b>break</b> <b>continue</b>	Interrupción de un bucle Continuación de un bucle

Estructuras de control de flujo dentro de Python

Fuente: Coremsa

- **Definición de funciones**, para definir una función en Python utilizamos la siguiente sintaxis:

```
def nombre_funcion(parametros):
```

```
    #comentarios
```

```
    Acciones
```

```
    return resultado
```

- **Parámetros**: sucesión de identificadores separados por comas que se le pasan a la función para que realice su menester.
- **return**: devuelve el resultado de realizar la función con los parámetros pasados. Si no se especifica, la función devuelve None y en este caso se denomina procedimiento.

A partir de la anterior definición podríamos tener el siguiente ejemplo:

```
def restaComprobada (a=None, b=None):
```

```
    if a == None or b == None:
```

```
        print "Error, debes enviar dos números a la función"
```

```
        return None
```

```
    else
```

```
        return a - b
```



### COMPRUEBA LO QUE SABES

Acabamos de realizar un repaso de las principales características de Python.

¿Serías capaz de saber el uso de una clase dentro de Python?

Busca un ejemplo en la red que ejemplarice tus comentarios.

Coméntalo en el foro.

## 2.3. Formularios e informes en sistemas ERP-CRM

La generación de nuevos formularios e informes pasa por conocer cómo funcionan los módulos dentro de Odoo, y para ello la construcción de un nuevo módulo en Odoo es una buena práctica para conocer todos los aspectos. Tanto el servidor como las extensiones de cliente en Odoo se empaquetan en módulos que a su vez se carga a través de la base de datos.

Las principales partes de un módulo en Odoo son:

- **Objetos de negocio o lógica de negocio.** Se declaran como clases de Python y son recursos que se hacen persistentes (se cargan) a través de configuración.
- **Vistas.** Donde se definen los interfaces con el usuario.
- **Ficheros de configuración y datos.** Son ficheros XML o CSV que definen metadatos sobre las vistas, los informes, la configuración y otros. Los hemos utilizado en unidades anteriores para modificar vistas, por ejemplo.
- **Controladores web.** Declaran las rutas que el navegador puede realizar.
- **Datos estáticos** como imágenes, css o javascript usados para diferentes puntos del cliente/servidor.

### 2.3.1. Llamadas a funciones, librerías de funciones

Como comentábamos al indicar las características de Python, una de las principales ventajas de este lenguaje es la multitud de funciones que tienes disponibles en forma de bibliotecas, o como se llaman en Python “módulos”. Si queremos acceder a una función que se encuentra en una de las bibliotecas de Python debemos seguir los siguientes pasos:

- 1) Importar la biblioteca:

**import nombre\_biblioteca** Ejemplo: `import math`

- 2) Acceder a la función cuando la necesitemos tan solo escribiendo el nombre de esta precedido por el nombre de la biblioteca en la que se encuentra y separados por un punto.

**Nombre\_biblioteca.nombre\_funcion(parametros)** Ejemplo:  
`print math.sin(2.0)`

Una alternativa al método expuesto anteriormente sería el importar de esta cada vez que queramos utilizar una de sus funciones.

**From nombre\_biblioteca import\*** Ejemplo: `from math import *`

También es posible importar varias librerías a la vez en una sola sentencia tan solo separándolas por comas.

Tal y como veremos durante la creación de los módulos, odoo utiliza una serie de paquetes que deben importarse en el entorno para su correcto funcionamiento. Usaremos la herramienta de Python pip para realizar esta tarea:

```
pip install Jinja2
```



#### ARTÍCULO DE INTERÉS

En el siguiente artículo tenemos un resumen del uso de pip.

<https://programminghistorian.org/es/lecciones/instalar-modulos-python-pip>

### 2.3.2. Elementos principales

Cada módulo es un directorio dentro del directorio principal de módulos. Un módulo tiene dos ficheros principales de definición:

- El *manifest*
- El fichero de importaciones o paquetes de Python definido dentro del fichero `__init__.py`

Para la creación de un nuevo módulo usaremos el siguiente comando:

```
odoo-bin scaffold <nombre del módulo> <subdirectorio>
```

Para nuestro ejemplo, en el caso de Windows 10, y desde el directorio `C:\Program Files (x86)\Odoo 13.0\server`, ejecutaremos:

```
./odoo-bin scaffold mimodulo addons
```

Puede que durante el uso de `odoo-bin`, aparezca el siguiente error:

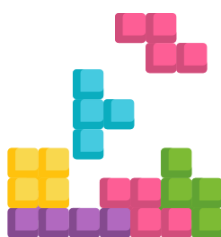
```
pacog@DESKTOP-FMSL9H7 MINGW64 /c/Program Files (x86)/Odoo 13.0
$ ./server/odoo-bin scaffold menu addons
Traceback (most recent call last):
  File "./server/odoo-bin", line 5, in <module>
    import odoo
  File "C:\Program Files (x86)\Odoo 13.0\server\odoo\__init__.py", line 132, in <module>
    from . import cli
  File "C:\Program Files (x86)\Odoo 13.0\server\odoo\cli\__init__.py", line 11, in <module>
    from . import scaffold
  File "C:\Program Files (x86)\Odoo 13.0\server\odoo\cli\scaffold.py", line 9, in <module>
    import jinja2
ModuleNotFoundError: No module named 'jinja2'

pacog@DESKTOP-FMSL9H7 MINGW64 /c/Program Files (x86)/Odoo 13.0
$
```

Error por falta de paquetes  
Fuente: imagen propia

Tal y como hemos visto en el apartado anterior, deberíamos usar `pip` para la instalación de los diferentes paquetes necesarios para usar `odoo-bin`:

Instalar todos los paquetes requeridos a partir del fichero de requirements
<code>pip install -r ./server/requirements.txt</code>
Instalar un paquete
<code>pip install Jinja2</code>



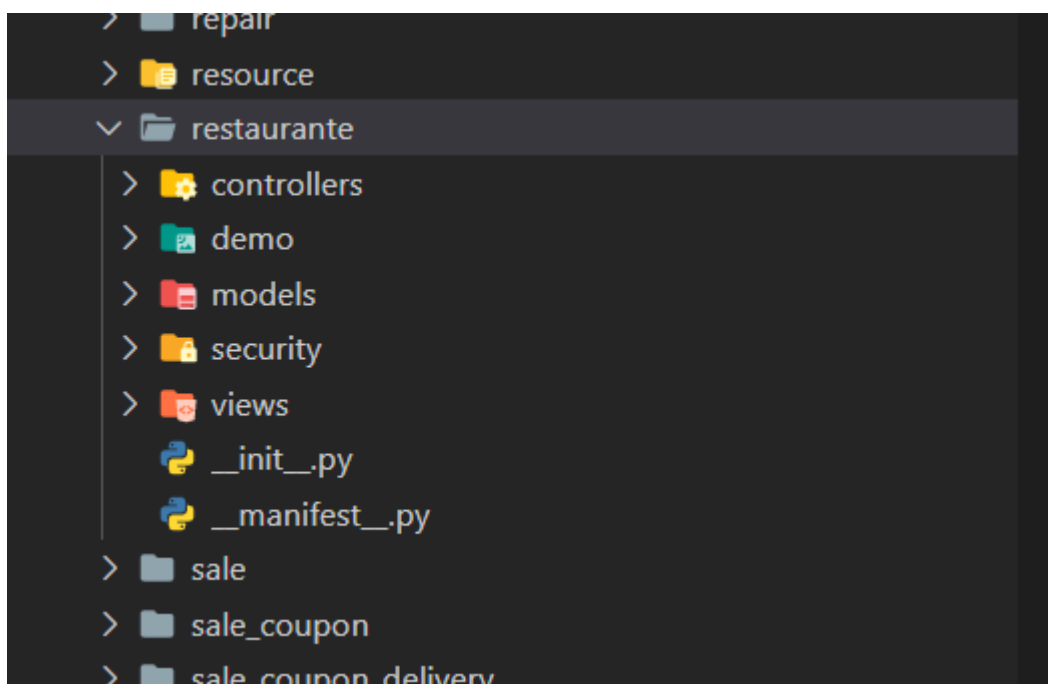
## EJEMPLO PRÁCTICO

Como desarrollador de software dentro de una cadena de restaurantes somos los encargados del mantenimiento y evolución del software de gestión empresarial instalado en nuestra empresa, en este caso Odoo v13.

Queremos realizar unas pruebas para añadir un módulo de platos de menú dentro de nuestra aplicación que nos permitan realizar justamente listados de productos que en nuestro caso son los platos.

Para ello seguiremos los siguientes pasos:

- 1) Usaremos `./server/odoo-bin scaffold restaurante "C:\Program Files (x86)\Odoo 13.0\server\odoo\addons"`, para crear el nuevo módulo



Módulo creado menús

Fuente: imagen propia

- 2) Adaptamos el fichero `__manifest__.py`, modificando los apartados:
  - a. summary
  - b. description
  - c. author
  - d. website
  - e. category

```
# -*- coding: utf-8 -*-
{
    'name': "restaurante",

    'summary': ""Modulo de platos y menus"",

    'description': ""
        Módulo para manejar:
        - platos
        - menus
    """,

    'author': "Paco Gomez",
    'website': "http://www.repositoriocompartido.com",

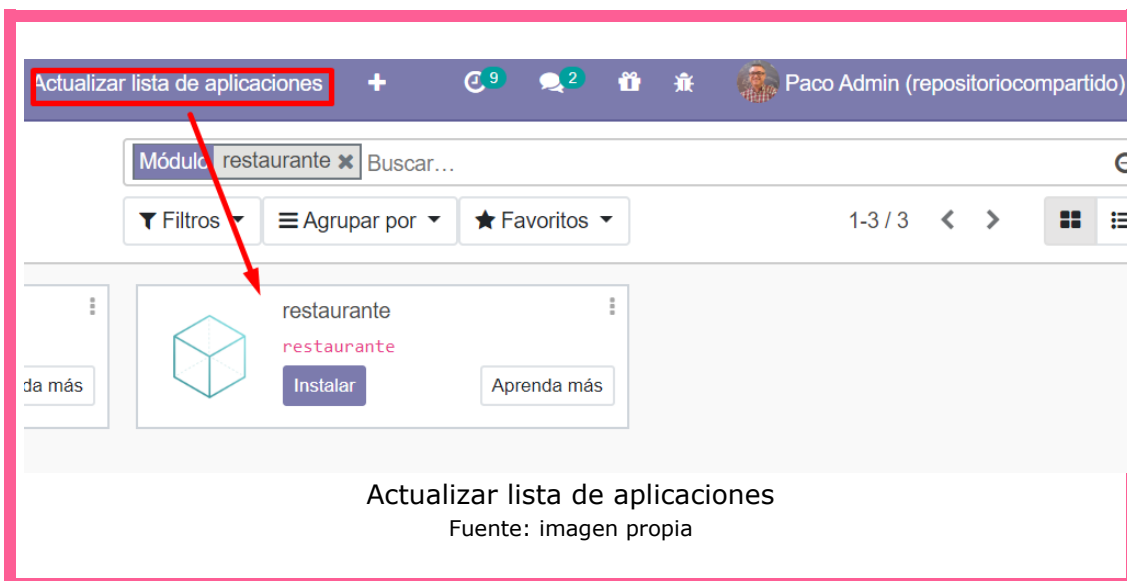
    # Categories can be used to filter modules in modules listing
    # Check https://github.com/odoo/odoo/blob/13.0/odoo/addons/base/data/i
r_module_category_data.xml
    # for the full list
    'category': 'Test',
    'version': '0.1',

    # any module necessary for this one to work correctly
    'depends': ['base'],

    # always loaded
    'data': [
        # 'security/ir.model.access.csv',
        'views/views.xml',
        'views/templates.xml',
    ],
    # only loaded in demonstration mode
    'demo': [
        'demo/demo.xml',
    ],
}
```

- 3) Para poder visualizar e instalar en el futuro el módulo como desarrolladores deberemos actualizar el listado de módulos en la pantalla de aplicaciones





### 2.3.3. Inserción, modificación y eliminación de datos

Mediante la capa **ORM** que proporciona Odoo, podemos crear modelos en nuestros módulos sin apenas escribir sentencias SQL y de esta forma aislar la lógica de la vista. Para ello declararemos clases que extenderán de **Model**, cuya clase automáticamente implementa la persistencia contra la base de datos.

Los modelos se pueden configurar con atributos, como cualquier clase, siendo **\_\_name** el único obligatorio e importante que define el nombre del modelo. A partir de la documentación oficial de Odoo tenemos una definición mínima de un modelo.

```
from odoo import models
class MinimalModel(models.Model):
    _name = 'test.model'
```

A partir de la definición de campo podemos definir una serie de características o atributos. Algunos atributos son:

- string, define la etiqueta en la vista.
- required, define si el campo puede estar vacío o no.
- help, define una ayuda para los usuarios en la vista.
- index, odoo crea un índice para esta columna.

La anterior clase podría ser modificada de la siguiente forma con el atributo required.

```
from odoo import models
class MinimalModel(models.Model):
    _name = 'test.model'

    name = field.Char(required=True)
```

Una vez que ya tenemos creado un modelo, podemos añadir datos de ejemplo que además ejemplarizarán el resto de las componentes visibles. Para ello utilizaremos el lenguaje de marcas XML y el elemento **<record>**. Cada elemento <record> creará o actualizará los elementos de la base de datos. Un ejemplo lo tenemos en el siguiente código:

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

</odoo>
```

Donde tenemos:

- model, el nombre del modelo para la entrada.
- id, es el identificador externo.
- field, corresponde a los campos dentro de la definición del modelo.

El contenido de los datos de demo se actualizará en la base de datos cuando instalemos o actualicemos el módulo.



### COMPRUEBA LO QUE SABES

Acabamos de introducir los conceptos sobre módulos y modelos.

¿Para qué sirve un modelo, es lo mismo que un módulo? Busca un ejemplo en la red que ejemplarice tus comentarios.

Coméntalo en el foro.

### 2.3.4. Acciones y menús

Las acciones y los menús nos permitirán poder navegar a través de las diferentes vistas del módulo o realizar tareas. Las acciones pueden ser activadas de tres formas:

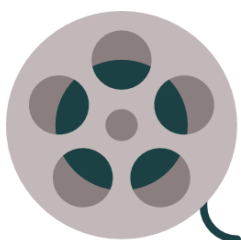
- Al hacer clic sobre un ítem de menú
- Al hacer clic sobre un botón en las vistas
- Como acciones contextuales sobre un objeto

Un ejemplo de menú y una acción dentro del menú lo tenemos en la siguiente definición:

```
<record model="ir.actions.act_window" id="action_list_ideas">
  <field name="name">Ideas</field>
  <field name="res_model">idea.idea</field>
  <field name="view_mode">tree,form</field>
</record>
```

Como se observa, se utiliza el objeto `ir.actions.act_window` para incluir una nueva acción dentro de la base de datos, que en este caso abrirá un formulario. Una vez definido, se puede utilizar esta acción dentro del menú. El código completo sería:

```
<record model="ir.actions.act_window" id="action_list_ideas">
  <field name="name">Ideas</field>
  <field name="res_model">idea.idea</field>
  <field name="view_mode">tree,form</field>
</record>
<menuitem id="menu_ideas" parent="menu_root" name="Ideas" sequence="10"
  action="action_list_ideas"/>
```



#### VIDEO DE INTERÉS

A continuación se muestra un vídeo muy detallado de la creación de un menú personalizado.

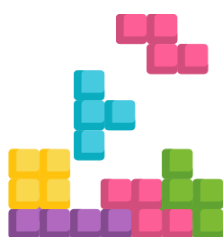
<https://www.youtube.com/watch?v=OB5hwe1o6cY>

### 2.3.5. Seguridad

La seguridad nos permite implementar los derechos de acceso a las diferentes partes de nuestro software y por lo tanto nos permite definir los accesos en nuestros módulos.

Los derechos de acceso se definen a partir del modelo **ir.model.access**. Cada propiedad de acceso está asociada a un modelo, grupo (puede que tenga grupo específico, y si no es así se asocia al acceso global), y por último los permisos de acceso (lectura, escritura, creación y desconexión). Esos accesos se suelen crear en un fichero csv. Un ejemplo de ese fichero puede ser como el de la imagen:

```
id,name,model_id/id,group_id/id,perm_read,perm_write,perm_create,perm_unlink
access_idea_idea,idea.idea,model_idea_idea,base.group_user,1,1,1,0
access_idea_vote,idea.vote,model_idea_vote,base.group_user,1,1,1,0
```

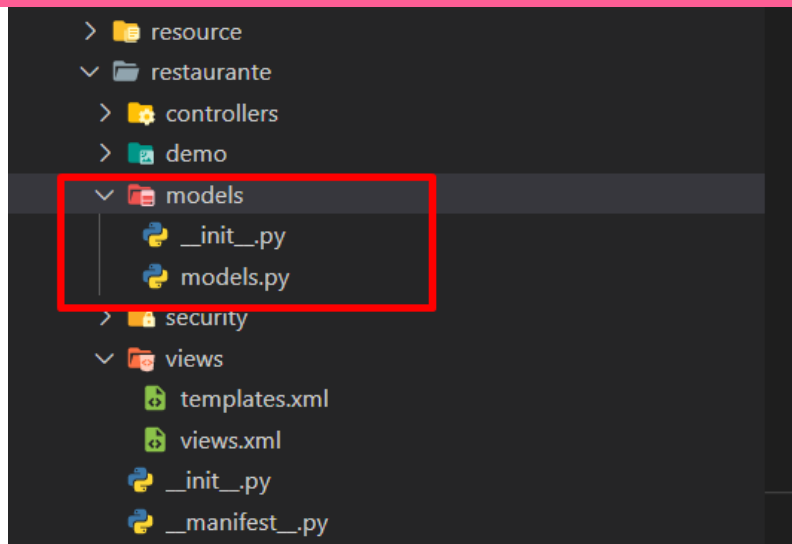


#### EJEMPLO PRÁCTICO

Como desarrollador de software dentro de una cadena de restaurantes somos los encargados del mantenimiento y evolución del software de gestión empresarial instalado en nuestra empresa, en este caso Odoo v13.

Tenemos un módulo recién creado mediante el comando **./server/odoo-bin scaffold menus "C:\Program Files (x86)\Odoo 13.0\server\odoo\addons"**, y queremos crear un nuevo modelo que será un plato que tendrá un título y una descripción, así como los datos de demo y un enlace en el menú para abrir tanto el listado de platos como el formulario de creación de un nuevo plato.

- 1) Definiremos un nuevo modelo Plato dentro del módulo restaurante. Este plato tendrá un título y una descripción. Para ello editaremos el fichero `menus/models/models.py`



Inclusión de nuevo modelo

Fuente: imagen propia

2) En el fichero realizaremos la siguiente modificación

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

# -*- coding: utf-8 -*-
from odoo import models, fields, api

class Plato(models.Model):
    _name = 'restaurante.plato'
    _description = "Plato de restaurante"

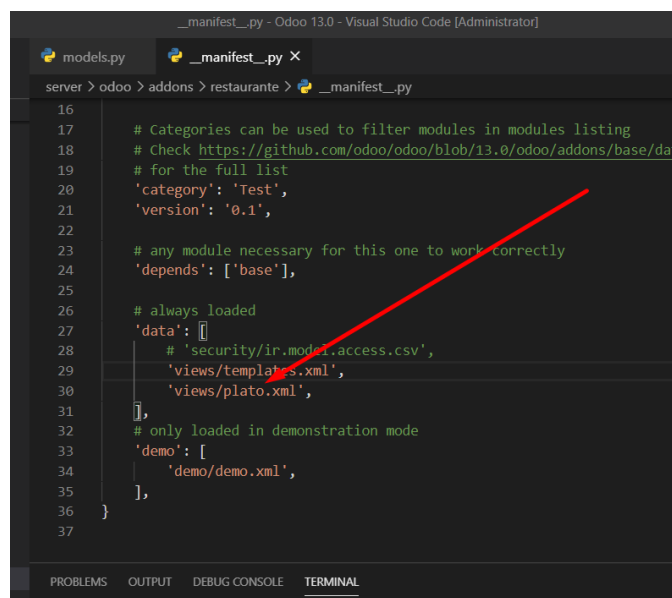
    name = fields.Char(string="Titulo", required=True)
    description = fields.Text()
```

3) El siguiente paso será crear datos de ejemplo para este nuevo modelo modificando **restaurante/demo/demo.xml**

```
<odoo>
<data>
  <record model="restaurante.plato" id="plato0">
    <field name="name">Plato 0</field>
    <field name="description">Descripcion del plato 0</field>
  </record>
  <record model="restaurante.plato" id="plato1">
    <field name="name">Plato 1</field>
    <!-- no description for this one -->
  </record>
  <record model="restaurante.plato" id="plato2">
    <field name="name">Plato 2</field>
```

```
<field name="description">Descripcion del plato 2</field>
</record>
</data>
</odoo>
```

- 4) Para añadir las nuevas acciones dentro del menú añadiremos una nueva vista que contendrá justamente los ítems de menú. Esto lo realizamos en primer lugar modificando el `__manifest__.py`



```
__manifest__.py - Odoo 13.0 - Visual Studio Code [Administrator]
models.py __manifest__.py X
server > odoo > addons > restaurante > __manifest__.py
16
17 # Categories can be used to filter modules in modules listing
18 # Check https://github.com/odoo/odoo/blob/13.0/odoo/addons/base/data
19 # for the full list
20 'category': 'Test',
21 'version': '0.1',
22
23 # any module necessary for this one to work correctly
24 'depends': ['base'],
25
26 # always loaded
27 'data': [
28     # 'security/ir.model.access.csv',
29     'views/templates.xml',
30     'views/plato.xml',
31 ],
32 # only loaded in demonstration mode
33 'demo': [
34     'demo/demo.xml',
35 ],
36 }
37
```

Modificación dentro de `__manifest__.py` para incluir las vistas

*Fuente: imagen propia*

- 5) Creamos el fichero `plato.xml` a partir de la documentación y el ejemplo de fabricante Odoo

```
<?xml version="1.0" encoding="UTF-8"?>
<odoo>

<!-- window action -->
<!--
    The following tag is an action definition for a "window action",
    that is an action opening a view or a set of views
-->
<record model="ir.actions.act_window" id="plato_list_action">
    <field name="name">Platos</field>
    <field name="res_model">restaurante.plato</field>
    <field name="view_mode">tree,form</field>
    <field name="help" type="html">
        <p class="o_view_nocontent_smiling_face">Crear el primer plato</p>
    </field>
</record>
```

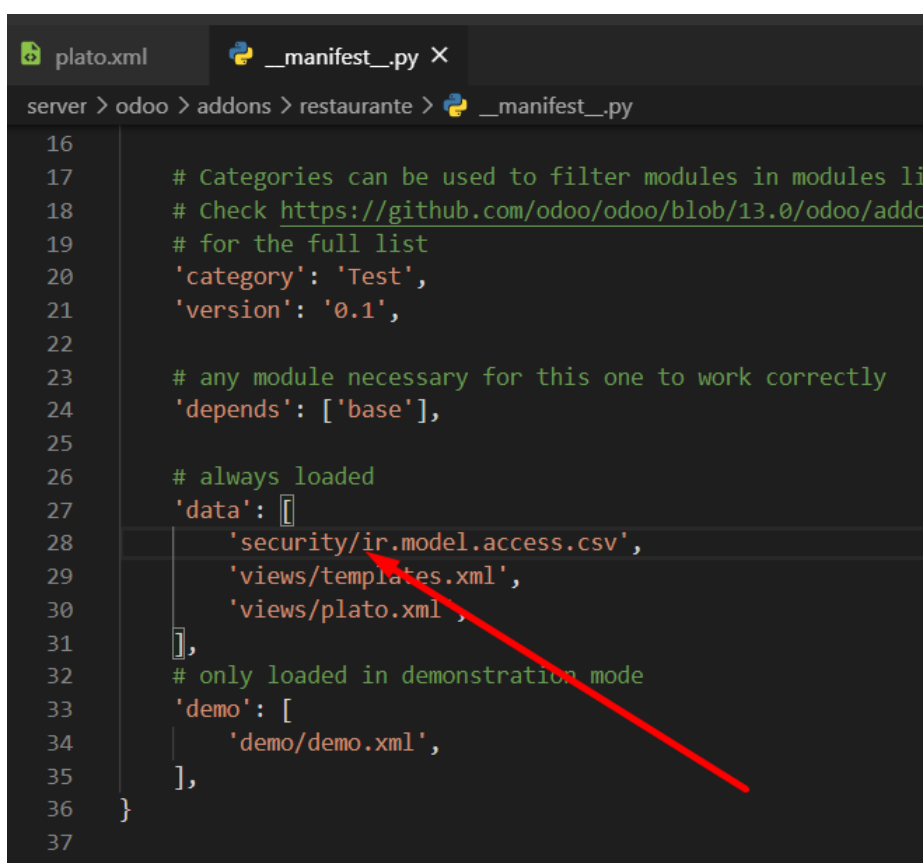
```

<!-- top level menu: no parent -->
<menuitem id="main_restaurante_menu" name="Restaurante"/>
<!-- A first level in the left side menu is needed
      before using action= attribute -->
<menuitem id="restaurante_menu" name="Restaurante"
      parent="main_restaurante_menu"/>
<!-- the following menuitem should appear *after*
      its parent restaurante_menu and *after* its
      action plato_list_action -->
<menuitem id="platos_menu" name="Platos" parent="restaurante_menu"
      action="plato_list_action"/>
<!-- Full id location:
      action="restaurante.plato_list_action"
      It is not required when it is the same module -->

</odoo>

```

6) Por último, crearemos los derechos de acceso a la aplicación.



```

server > odoo > addons > restaurante > __manifest__.py
16
17     # Categories can be used to filter modules in modules li
18     # Check https://github.com/odoo/odoo/blob/13.0/odoo/addons
19     # for the full list
20     'category': 'Test',
21     'version': '0.1',
22
23     # any module necessary for this one to work correctly
24     'depends': ['base'],
25
26     # always loaded
27     'data': [
28         'security/ir.model.access.csv',
29         'views/templates.xml',
30         'views/plato.xml',
31     ],
32     # only loaded in demonstration mode
33     'demo': [
34         'demo/demo.xml',
35     ],
36 }
37

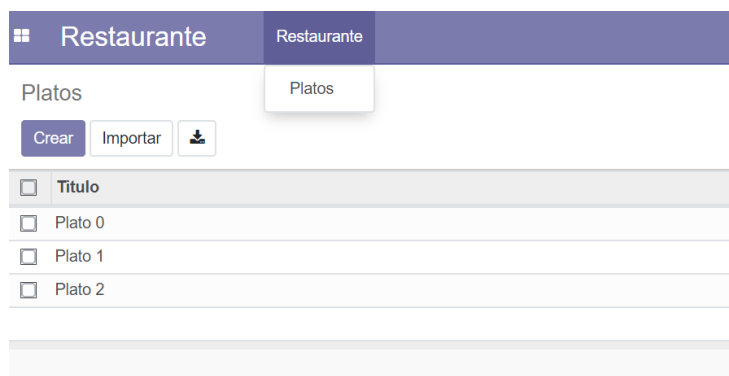
```

Modificación dentro de \_\_manifest\_\_.py para incluir las seguridad

Fuente: imagen propia

```
id,name,model_id:id,group_id:id,perm_read,perm_write,perm_create,perm_u
nlink
session_manager,session
manager,model_restaurante_plato,sales_team.group_sale_manager,1,1,1,1
session_read_all,session all,model_restaurante_plato,1,0,0,0
```

7) Instalamos el módulo y comprobamos el funcionamiento



Modificación dentro de `__manifest__.py` para incluir las seguridad  
Fuente: imagen propia

## 2.4. Extracciones de informaciones contenidas en sistemas ERP-CRM

Las vistas definen los mecanismos con los que un modelo en concreto es visualizado y por lo tanto sobre el que podemos realizar consultas. Cada tipo de vista representa un tipo de visualización (un listado de campos, un gráfico, un formulario, ...). Aunque no es el ámbito de profundidad de este curso, las vistas dentro de Odoo permiten la herencia, por lo que pueden ser incluidos partes de vistas ya creados en otros módulos.

Tal y como hemos visto anteriormente, las vistas pueden ser llamadas a través de los menús de las aplicaciones y módulos, y a través de diferentes acciones gracias a su id.

Veremos los siguientes diseños de vistas o informes básicos:

- Listados de registros
- Formularios
- Gráficos



### 2.4.1. Herramientas para la creación de formularios e informes

De igual forma que estamos trabajando la creación de un nuevo módulo dentro de Odoo, tenemos herramientas para generar formularios, informes, gráficos y vistas en general.

Cualquier formulario, informe o vista es declarado como una nueva entrada del Modelo dentro de Odoo denominado ***ir.ui.view***. Veamos un ejemplo de código extraído de la propia documentación de Odoo:

```
<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>
```

Dentro del ejemplo tenemos las siguientes partes:

- **Elemento <record>** que permite definir un nuevo elemento en el modelo marcado por el atributo en cuestión.
- **Atributo model** permite indicar cual es el modelo donde se incluye la nueva entrada, en nuestro caso el modelo es *ir.ui.view*, que tal y como hemos dicho almacena en la base de datos las diferentes vistas.
- **Elemento <field>** permite definir los diferentes campos propios del modelo que se esté trabajando, en nuestro caso *ir.ui.view*:
  - Campo **name**, define el nombre de la vista.
  - Campo **model**, define el modelo de la vista sobre la que aplica. No confundir con el modelo *ir.ui.view*, que es el modelo que almacena los id de las vistas.
  - Campo **arch**, define el tipo de documento, en nuestro caso XML.



### COMPRUEBA LO QUE SABES

Acabamos de introducir la realización de una vista básica.

¿Serías capaz de distinguir sobre el uso de `<record>` y `<field>` ? Busca un ejemplo en la red que ejemplarice tus comentarios.

Coméntalo en el foro.

### 2.4.2. Diseño de informes

Un informe puede definirse de múltiples formas y visualizaciones veamos algunos ejemplos de definiciones de estos informes.

- **Tree View**, en el que se muestran los campos de forma tabular. Es el informe más sencillo ya que cada registro se muestra en una fila y cada campo en una columna. Como vemos en el siguiente ejemplo, el elemento xml raíz es **`<tree>`**

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

- **Formulario**, se utilizan para crear nuevas entradas dentro de los modelos definidos. Como vemos en el siguiente ejemplo, el elemento raíz es `<form>`, e internamente se compone de una estructura de grupos, botones, campos, etc., dependiendo de la complejidad del formulario.

```
<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>
```

```
<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>
```

- **Gráficos**, se utilizan para diseñar agregados, análisis y representaciones gráficas de los datos dentro de un modelo. Como vemos en el siguiente ejemplo, el elemento raíz es <graph> y permite definir tipos de visualizaciones dependiendo del atributo type (barras, línea, tarta, ...)

```
<graph string="Total idea score by Inventor">
  <field name="inventor_id"/>
  <field name="score" type="measure"/>
</graph>
```

## 2.5. Operaciones de consulta. Herramientas

Hemos visto como dentro de un módulo de Odoo implementamos los diferentes modelos que nos permiten guardar y extraer información de forma persistente desde la base de datos.

Un modelo puede necesitar datos de otra aplicación, módulo o de otro modelo. Por ejemplo, los pedidos de venta están relacionados con los datos del cliente.

Para poder relacionar datos de dos módulos tenemos las siguientes relaciones:

- **many2one**, en este caso tenemos la relación entre dos modelos de 1 a n elementos. La sintaxis extraída de la documentación de fabricante:

```
many2one(other_model, ondelete='set null')
```

- **one2many**, en este caso tenemos la relación entre dos modelos de n a 1 elementos. En este caso para que se pueda tener una relación one2many en un modelo, necesitamos que exista una relación many2one en el primer modelo. La sintaxis extraída de la documentación de fabricante:

```
one2many(other_model, related_field)
```

- **many2many**, en este caso tenemos la relación bidireccional entre dos modelos de n a n elementos.



### COMPRUEBA LO QUE SABES

Acabamos de introducir relación entre campos de los modelos.

¿Serías capaz de diferenciar entre many2one y one2many ?  
Pon un ejemplo que pueda identificar esas diferencias.

Coméntalo en el foro.



## EJEMPLO PRÁCTICO

Como desarrollador de software dentro de una cadena de restaurantes somos los encargados del mantenimiento y evolución del software de gestión empresarial instalado en nuestra empresa, en este caso Odoo v13.

Tenemos un módulo recién creado mediante el comando `./server/odoo-bin scaffold menus "C:\Program Files (x86)\Odoo 13.0\server\odoo\addons"`, en el que hemos desarrollado un modelo de platos. Necesitamos poder crear nuevos menús con una fecha de inicio y una duración del menú, un responsable del menú que será un usuario y además añadir al enlace tanto el listado como el nuevo formulario:

- 1) Crearemos un nuevo modelo para tener el Menu que comienza a estar activo en una fecha, por un período de tiempo y con unas calorías. El fichero models.py queda de la siguiente forma:

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

from odoo import models, fields, api

class Plato(models.Model):
    _name = "restaurante.plato"
    _description = "Plato de restaurante"

    name = fields.Char(string="Title", required=True)
    description = fields.Text()

class Carta(models.Model):
    _name = 'restaurante.carta'
    _description = "Menu de restaurante"

    name = fields.Char(string="Title", required=True)
    description = fields.Text()
    start_date = fields.Date()
    duration = fields.Float(digits=(6, 2), help="Activo en días")
    calorías = fields.Integer(string="Calorías")
```

- 1) Añadiremos el responsable de la carta usando many2one dentro del modelo carta:

```
responsable_id = fields.Many2one('res.users',
    ondelete='set null', string="Responsable", index=True)
```

- 2) Renombraremos plato.xml a restaurante.xml para una mejor comprensión y añadiremos en primer lugar la vista:

```
<record model="ir.actions.act_window" id="menu_list_action">
  <field name="name">MEnus</field>
  <field name="res_model">restaurante.carta</field>
  <field name="view_mode">tree,form</field>
</record>

<menuitem id="carta_menu" name="Menus"
  parent="restaurante_menu"
  action="menu_list_action" sequence="2"
/>
```

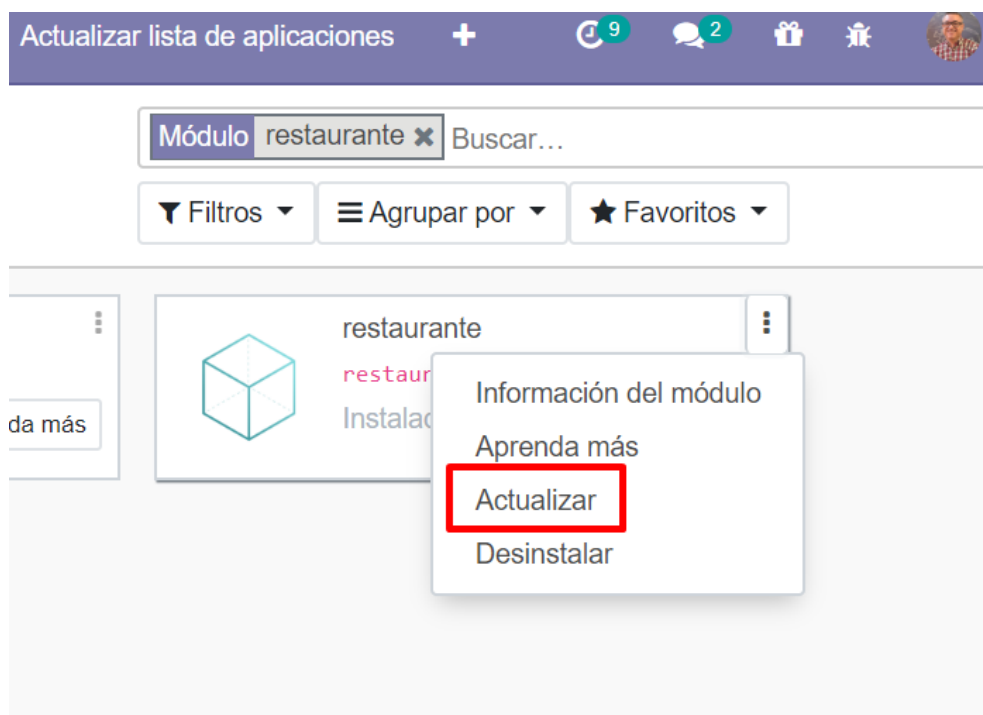
- 3) Y en segundo lugar un listado personalizado:

```
<!-- session form view -->
<record model="ir.ui.view" id="carta_form_view">
  <field name="name">carta.form</field>
  <field name="model">restaurante.carta</field>
  <field name="arch" type="xml">
    <form string="Menu Form">
      <sheet>
        <group>
          <field name="name"/>
          <field name="start_date"/>
          <field name="duration"/>
          <field name="calorias"/>
          <field name="responsible_id"/>
        </group>
      </sheet>
    </form>
  </field>
</record>
```

- 4) Añadimos las configuraciones de seguridad.

```
id,name,model_id:id,group_id:id,perm_read,perm_write,perm_create,perm_u
nlink
session_manager,session
manager,model_restaurante_plato,sales_team.group_sale_manager,1,1,1,1
session_manager,session
manager,model_restaurante_carta,sales_team.group_sale_manager,1,1,1,1
session_read_all,session all,model_restaurante_carta,1,0,0,0
```

5) Actualizamos el módulo:



Actualización del módulo  
Fuente: imagen propia

### ***2.5.1. Técnicas de optimización de consultas y acceso a grandes volúmenes de información***

Cuando se realizan consultas a grandes bases de datos, se nos presenta la pregunta de cómo poder optimizar las consultas para que el rendimiento del sistema mejore o sea el óptimo.

Para comenzar debemos seguir dos grandes consejos siempre:

- En el caso de que seamos propietarios y administradores de la base de datos, es nuestro caso con Odoo, se debe intentar realizar procedimientos almacenados en la base de datos. Un **procedimiento almacenado** consiste en una consulta o conjunto de funciones que se ejecutan en la base de datos y que al estar dentro de la misma base de datos siempre será mucho más rápida que ser lanzada desde una aplicación externa.

- Generar **vistas** dentro de la base de datos PostgreSQL. Es otro mecanismo dentro de la propia base de datos que nos permite agrupar la información para poderla alcanzar de una forma rápida, ya que normalmente esa información procede de varias tablas.
- Por último, intentar realizar el mínimo número de consultas posibles desde la aplicación. Aquí también existen muchas técnicas, que han evolucionado en la actualidad y que permiten mantener una serie de información en el lado del cliente sin necesidad de ser actualizada.

Evidentemente, la combinación de varias de las técnicas descritas anteriormente es posible y necesaria. Normalmente se siguen las siguientes premisas:

- Realizar procedimientos almacenados y vistas en el caso de consultas que se conoce su parámetros de entrada y que por lo tanto no dependen de los datos que proporcione el usuario para formar la información.
- Almacenar en cache los datos que son usados en la sesión o datos cuyo valor no cambia durante la sesión de trabajo.
- Recoger el máximo posible de información en una consulta para ser tratada posteriormente al estar ya almacenada en memoria.

Como podemos observar, la máxima, es intentar no realizar muchas consultas a la base de datos, ya que este medio, es el más lento e ineficiente dentro de la cadena.



#### ARTÍCULO DE INTERÉS

En el siguiente artículo dispondremos de más información sobre los procedimientos almacenados.

<https://e-mc2.net/es/procedimientos-almacenados-y-plpgsql>



### **2.5.2. Generación de programas de extracción de datos entre sistemas (batch inputs)**

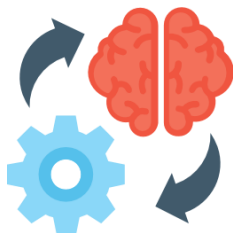
La extracción de datos entre sistemas, pasarelas o adaptadores entre software de gestión empresarial son procesos importantes y comunes dentro de las empresas.

En una empresa nos podemos encontrar con diferentes software de gestión de la información, bien porque se han realizado compras e implantaciones de software específico para las necesidades de departamentos, bien porque el software ha ido evolucionando. La extracción e importación (batch inputs) suelen ser procesos muy habituales programados para realizar alguna de las siguientes tareas:

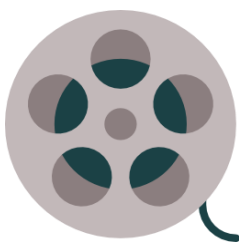
- Extracción automática de información para la generación de informes, suele producirse cuando se usan sistemas de BI, o generadores de cuadros de mandos, donde se suele utilizar bases de datos intermedias que preparan la información para poder después realizar los informes.
- Extracción automática de la información para el uso en un software externo. Es un caso muy parecido al anterior, solo que en este caso los datos se suelen utilizar en un segundo software de gestión empresarial que generará nueva información a partir de los datos del primer software.
- Importación automática o batch input, en este caso nuestro software de gestión empresarial necesita de unos datos que provienen de software externos para que los usuarios puedan realizar sus tareas.

La programación de estos programas pasarela, suele ser a través de *scripts* externos que trabajan directamente con la base de datos y que tienen las siguientes características:

- Realizan la extracción de la información de la base de datos mediante secuencias SELECT.
- Realizan la importación de la información en otra base de datos mediante secuencias UPDATE e INSERT.
- Suelen ser *scripts* programados y automáticos en momentos temporales donde no haya trabajo o carga con los sistemas.

**RECUERDA**

En la UD4 en el punto 2 vimos cómo se realizaba el proceso de extracción de información a través de JasperStudio, generando informes particularizados.

**VIDEO DE INTERÉS**

En el siguiente vídeo podrás ver un buen ejemplo sobre automatización de procesos.

[https://www.youtube.com/watch?v=O7VTJKFZpe8&list=PLHgpVrCyLWAr sjTq399z7\\_M7KWTq4DFeH&index=30](https://www.youtube.com/watch?v=O7VTJKFZpe8&list=PLHgpVrCyLWAr sjTq399z7_M7KWTq4DFeH&index=30)

## 2.6. Depuración de un programa

Depurar un programa consiste en encontrar y corregir los problemas que pueda tener el código del mismo.

Podemos hacerlo por métodos “arcaicos” usando una herramienta básica como es la función `print()` para ir viendo los valores de las variables relevantes del programa, o ir imprimiendo texto por pantalla cada cierto número de instrucciones (sobre todo cuando las instrucciones llevan a una bifurcación como un `if else`) para saber por qué ramas del código transcurre la ejecución y qué valores van tomando las diferentes variables.

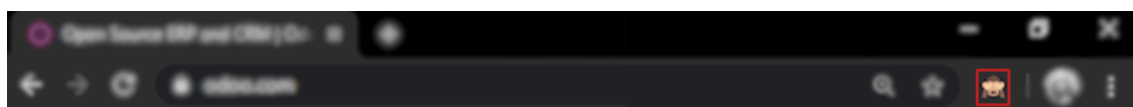
También contamos con herramientas más poderosas como es el Depurador, que nos ayuda a detectar y corregir los fallos más rápidamente al permitir la ejecución interactiva del programa. Como inconveniente del Depurador, hay que indicar que ralentiza la ejecución del código haciendo que un simple programa pueda durar varios minutos, pero hay técnicas para minimizar este punto. Utilizarlo nos proporciona una información muy útil que compensa con creces este inconveniente, hay errores que son muy difíciles de detectar y corregir con el uso exclusivamente de la función `print()`.

Odoo es un software que se ejecuta en modo cliente/servidor y podemos usar diferentes mecanismos para poder realizar un debugging de la aplicación:

- A través de mecanismos print con Python
- A través de la librería de log
- A través de plugins de navegador

Este último mecanismo es muy interesante ya que nos permite añadir un recubrimiento sin realizar cambios en el código fuente. Para ello seguiremos los siguientes pasos:

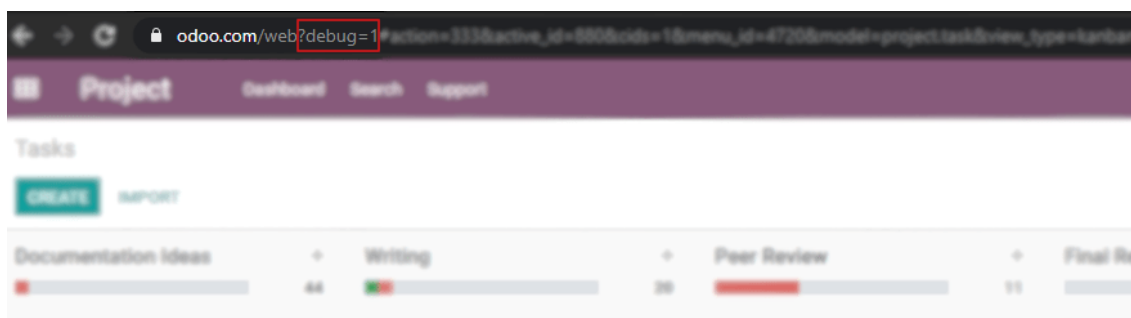
- 1) Instalaremos el plugin de Chrome, Odoo Debug



Activación del Odoo Debug en Chrome

Fuente: [https://www.odoo.com/documentation/user/13.0/es/general/developer\\_mode/activate.html](https://www.odoo.com/documentation/user/13.0/es/general/developer_mode/activate.html)

- 2) Podremos usar el debug añadiendo a la url de cada una de las páginas en modo desarrollador `?debug=1` o `?debug=true`



Activación a través de URL

Fuente: [https://www.odoo.com/documentation/user/13.0/es/general/developer\\_mode/activate.html](https://www.odoo.com/documentation/user/13.0/es/general/developer_mode/activate.html)



### ENLACE DE INTERÉS

A través del siguiente enlace podrás instalar el plugin de Debug sobre Odoo.

<https://chrome.google.com/webstore/detail/odoo-debug/hmdmhilocobgohohpdpolmibjklfgkbi>



### COMPRUEBA LO QUE SABES

Acabamos de estudiar los conceptos básicos sobre depuración.

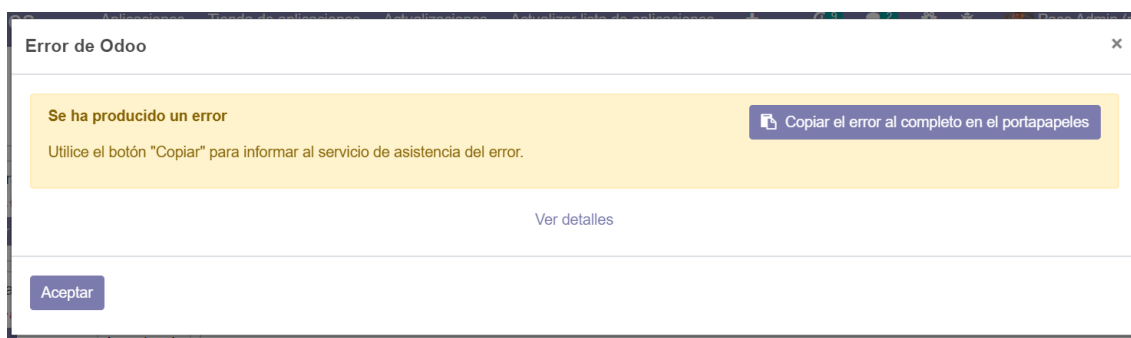
¿Serías capaz de distinguir entre una depuración en cliente y en servidor? ¿Qué recursos y herramientas necesitas para realizar una u otra?

Coméntalo en el foro.

### 2.6.1. Manejo de errores

Otra de las utilidades que tenemos con odoo en modo desarrollador es la de poder recibir errores cuando estos se producen durante el desarrollo. Errores que suelen ser muy comunes.

Cuando desarrollamos un módulo, durante su instalación o durante su actualización pueden aparecer ventanas de error como las que vemos en la imagen:



Ventana de error

Fuente: [https://www.odoo.com/documentation/user/13.0/es/general/developer\\_mode/activate.html](https://www.odoo.com/documentation/user/13.0/es/general/developer_mode/activate.html)

Cuando abrimos los detalles de esta ventana solemos encontrar 3 partes bien diferenciadas:

- Una primera parte extensa y detallada de todo el itinerario de los errores producidos dentro de la aplicación.

[Ver detalles](#)

```

Odoo Server Error

Traceback (most recent call last):
  File "C:\Program Files (x86)\Odoo 13.0\server\odoo\addons\base\models\ir_ui_view.py", line 394, in _check_xml
    self.postprocess_and_fields(view.model, view.doc, view.id)
  File "C:\Program Files (x86)\Odoo 13.0\server\odoo\addons\base\models\ir_ui_view.py", line 987, in postprocess_and_fields
    self.raise_view_error(message, view_id)
  File "C:\Program Files (x86)\Odoo 13.0\server\odoo\addons\base\models\ir_ui_view.py", line 614, in raise_view_error
    raise ValueError(message)
ValueError: El campo 'responsible_id' no existe

Contexto del error:
Vista 'carta.form'
[view_id: 934, xml_id: gestorRestaurantes.carta_form_view, model: restaurante.carta, parent_id: n/a]

```

### Traceback de los errores

Fuente: [https://www.odoo.com/documentation/user/13.0/es/general/developer\\_mode/activate.html](https://www.odoo.com/documentation/user/13.0/es/general/developer_mode/activate.html)

- Un mensaje específico del error concreto y donde se produce.
- Un contexto específico del error.

[Ver detalles](#)

```

File "C:\Program Files (x86)\Odoo 13.0\server\odoo\models.py", line 1176, in _validate_fields
    check(self)
File "C:\Program Files (x86)\Odoo 13.0\server\odoo\addons\base\models\ir_ui_view.py", line 396, in _check_xml
    raise ValidationError("%s\n%s" % (_("Error while validating view"), tools.ustr(e)))
odoo.tools.convert.ParseError: "Error while validating view

El campo 'responsible_id' no existe

Contexto del error:
Vista 'carta.form'
[view_id: 934, xml_id: gestorRestaurantes.carta_form_view, model: restaurante.carta, parent_id: n/a]
None" while parsing file:/c:/program%20files%20(x86)/odoo%2013.0/server/odoo/addons/gestorrestaurantes/views/restaurante.xml:2, near
<odoo>

<!-- window action -->

```

### Contexto del error

Fuente: [https://www.odoo.com/documentation/user/13.0/es/general/developer\\_mode/activate.html](https://www.odoo.com/documentation/user/13.0/es/general/developer_mode/activate.html)

A partir de esta información debemos corregir el error, y en este punto no existe una metodología única. El primer parámetro importante es la experiencia del desarrollador y la experiencia con el software lo que permitirá corregir los errores. Un pequeño proceso y metodología puede ser el siguiente:

- 1) Abrir el contexto que indica el mensaje del error, suele ser un fichero dentro del módulo.
- 2) Realizar cambios pequeños y comprobar si se ha resuelto el problema.
- 3) Si no se resuelve el problema de una forma corta y sencilla, realizar una vuelta atrás del código y comprobar el correcto funcionamiento.

## RESUMEN FINAL

Las empresas tienen necesidades diversas, que hacen que la implantación de un sistema de gestión empresarial ERP-CRM necesite en la gran mayoría de ocasiones de adaptaciones para poder dar respuesta a los diferentes procesos y operaciones que se realizan dentro del negocio.

Podemos entender el software empresarial desde tres puntos de vista: el software standard, el software híbrido y por último, el software a medida. Las técnicas y estándares que se utilizan en la adaptación y desarrollo a medida de software de gestión empresarial son las mismas que en el desarrollo de software. Se utilizan lenguajes de programación libre o código abierto, como es el caso de Odoo.

Una solución híbrida tal es una opción fantástica y muy válida para poder tener lo mejor de las soluciones a medida y de las soluciones estándar. Cuando una empresa o un negocio se plantea implantar un ERP-CRM realiza un proceso de estado del arte de este tipo de software, al mismo tiempo que realiza un estudio de las necesidades que transforma en unas especificaciones para el proyecto de implantación.

La generación de nuevos formularios e informes, pasa por conocer cómo funcionan los módulos dentro de Odoo, y para ello la construcción de un nuevo módulo en Odoo ya que es una buena práctica para conocer todos los aspectos: herramientas de desarrollo, modelos de datos y vistas.