



PROYECTO SGE

CFGS Desarrollo de Aplicaciones
Multiplataforma
Informática y Comunicaciones

**Desarrollo del módulo “Manage” con Odoo
ERP; para gestionar proyectos usando
metodologías ágiles: scrum**

Año: 2024

Fecha de presentación: 22/12/2024

Nombre y Apellidos: Erik Amo Toquero
Email: erik.amotoq@educa.jcyl.es

Contenido

1	Introducción.....	3
1.1	Objetivos del trabajo	3
1.2	Entorno de trabajo.....	3
2	Organización de la memoria	3
2.1	Instalación de Odoo	3
2.2	Modelo Relacional	8
2.3	Procedimiento y Pruebas de funcionamiento	8
2.3.1	Creación de los modelos.....	9
2.3.2	Dar acceso a la información.....	16
2.3.3	Creación de las vistas	16
2.3.4	Creación de las acciones	23
2.3.5	Orden de los menús.....	23
2.4	Ampliaciones.....	24
2.4.1	Relaciones	24
2.4.2	Computación de campos	29
2.4.3	<i>Extra: Agrupar Tasks por Sprint</i>	33
2.4.4	Herencia de desarrolladores con el módulo Contactos.....	34
3	Estado del arte	35
3.1	ERP	35
3.2	SCRUM	36
4	Conclusiones	37
5	Bibliografía	37

1 Introducción

Las empresas utilizan distintos sistemas para gestionar sus procesos y datos de la forma más eficiente posible. La unificación de esos sistemas se les llama **ERP** (*Enterprise Resource Planning*), y buscan controlar y modificar

1.1 Objetivos del trabajo

El objetivo principal es crear un módulo del ERP Odoo que permita manejar todas las partes de un proyecto utilizando la metodología SCRUM, pudiendo ver, añadir, editar y borrar los diferentes elementos. Dicho módulo tendrá cierta automatización, ya que las tablas estarán ligadas por medio del modelo EER de PostgreSQL y aparecerán también campos computados para facilitar el trabajo a quienes lo utilicen.

1.2 Entorno de trabajo

El entorno de trabajo es sencillo:

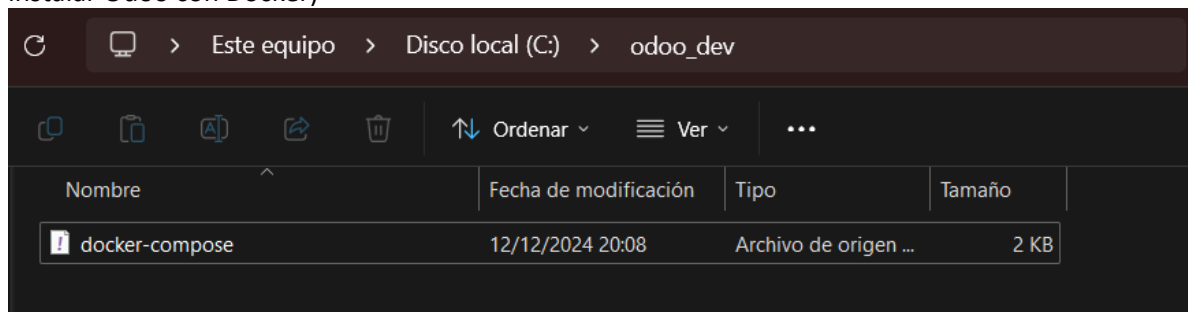
- Se utilizará un IDE que permita programar en Python, ya que en Odoo los módulos se programan en este lenguaje. El IDE que he elegido en mi caso es **Visual Studio Code**, ya que es un entorno que permite manipular varios lenguajes de forma sencilla y rápida, además de su inmensidad de extensiones para facilitar la programación.
- Para programar y probar un módulo de Odoo, hace falta tener el sistema de Odoo instalado. En este caso, para agilizar el proceso de programación y poder modificar el módulo en varios dispositivos con más facilidad, utilizaré **Docker**, una herramienta que permite descargar imágenes y crear contenedores, unos entornos virtuales que pueden ser ejecutados en el host. Es equiparable a la virtualización, salvo que comparte los recursos en tiempo real con la máquina física, así que es una alternativa al uso de máquinas virtuales más eficiente para este tipo de proyectos.

2 Organización de la memoria

A lo largo del desarrollo del proyecto, he ido siguiendo los distintos pasos para crear correctamente el módulo Manage de Odoo.

2.1 Instalación de Odoo

En mi caso, instalé Odoo por medio de un contenedor de Docker (TODO -> Meter info de como instalar Odoo con Docker)



Una vez esté el archivo yaml configurado a gusto del usuario, se crea e inicia el contenedor accediendo a la carpeta donde está el yaml con la terminal y utilizando el comando `docker-compose up -d`

```
PowerShell 7.4.6
PS C:\odoo_dev> docker-compose up -d
```

El programa habrá terminado de cargar en cuanto los 3 contenedores existentes (pgadmin, postgresdb y odoo) hayan sido creados e inicializados

```
PowerShell
PowerShell 7.4.6
PS C:\odoo_dev> docker-compose up -d
time="2024-12-12T20:10:49+01:00" level=warning msg="C:\\odoo_dev\\docker-compose.yaml: the attribute 'version' is obsolete, it will be ignored, please remove it to avoid potential confusion"
[+] Running 4/3
  ✓ pgadmin Pulled                                17.1s
  ✓ odoo Pulled                                    60.0s
  ✓ db Pulled                                     24.4s
[+] Running 6/6
  ✓ Network odoo_dev_red_odoo Created              0.8s
  ✓ Volume "odoo_dev_odoo-data" Created            0.0s
  ✓ Volume "odoo_dev_db-data" Created              0.0s
  ✓ Container container-postgresdb Started          0.9s
  ✓ Container odoo Started                         0.9s
  ✓ Container odoo_dev-pgadmin-1 Started            0.9s
PS C:\odoo_dev>
```

Información relevante del contenedor de Odoo:

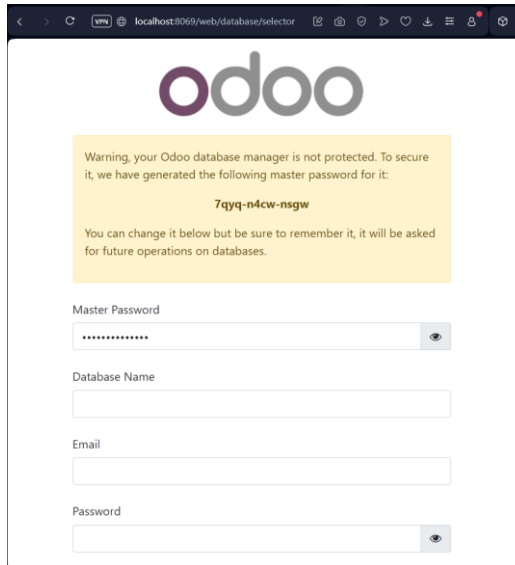
En el archivo yaml se guarda información de los controladores. Del contenedor de Odoo nos interesa lo siguiente, ya que se va a necesitar a futuro:

- image: Si al usuario le interesa cambiar la versión de Odoo, se hará cambiando el número del final (16, 17, *latest* para la última versión...)
- ports: Son los puertos que utiliza el contenedor para acceder a él. El primer puerto es el de la máquina física, mientras que el segundo es el puerto del contenedor.
- volumes: Son un "enlace" que se hace entre los ficheros del contenedor con la carpeta física del host. La primera parte es el directorio en la carpeta física y la segunda es la ubicación de sus ficheros en el contenedor
 - **config:** Guarda los datos de configuración de odoo
 - **addons:** Guarda los módulos del ERP

```
C: > odoo_dev > docker-compose.yaml
1  version: "3.8"
2  services:
3
4  odoo:
5    image: odoo:16
6    container_name: odoo
7    restart: unless-stopped
8    links:
9      - db:db
10   depends_on:
11     - db
12   ports:
13     - "8069:8069"
14   volumes:
15     - odoo-data:/var/lib/odoo
16     - ./config:/etc/odoo
17     - ./addons:/mnt/extra-addons
18   networks:
19     - red_odoo
20
```

Continuando con lo dicho previamente, para acceder al ERP una vez se inician los contenedores, es necesario escribir en un explorador de internet el puerto donde se aloja (localhost si se accede desde el host) y el **puerto que se ha elegido para la máquina local en el archivo yaml** (8069 en este caso)

La primera vez que se acceda al ERP, pedirá crear una cuenta. Aquí se tienen que introducir los datos que se crean necesarios.



En cuanto se crea el usuario y, con ello, su base de datos, saltará un menú de inicio de sesión. Se tendrán que escribir las credenciales puestas previamente.

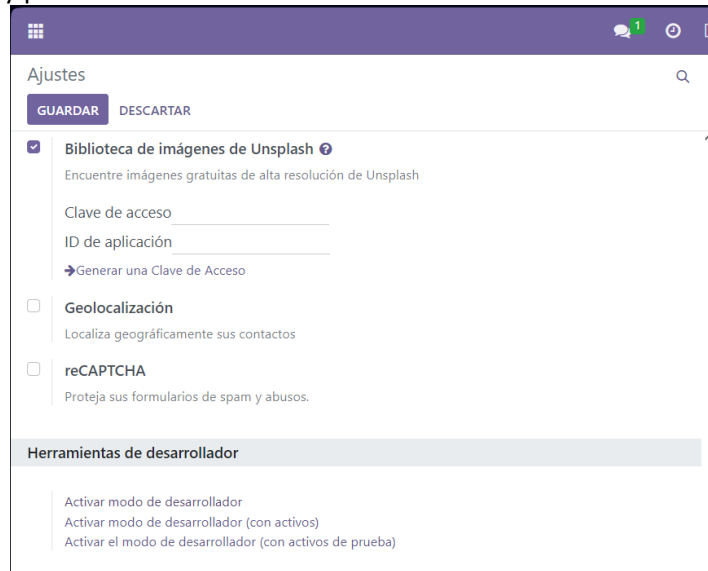


Lo siguiente a realizar en el ERP antes de crear el módulo es activar las opciones de desarrollador. Esto se consigue con los siguientes pasos:

1. **Activar un módulo que permita acceder a administrador una vez instalado.** Pueden servir los módulos CRM, Ventas o Finanzas.



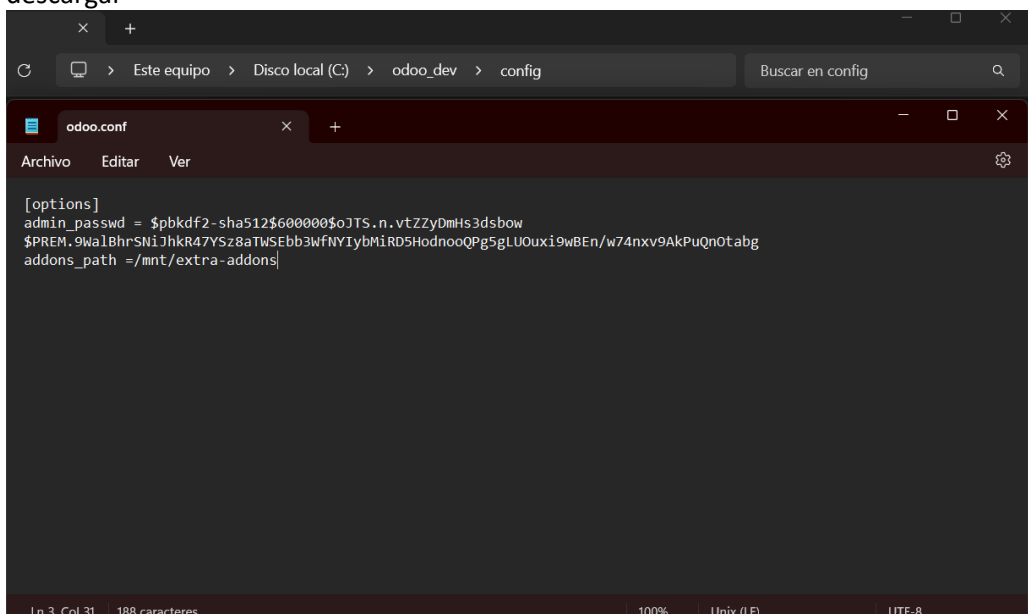
2. Ir a Ajustes y pulsar en "Activar modo de desarrollador"



Ahora que se ha activado el modo de desarrollador, se debe apagar el contenedor e ir a la carpeta donde se ha especificado en el archivo yaml que se guarden los archivos de configuración de Odoo. En este archivo, habrá que añadir la siguiente línea:

addons_path = /mnt/extra-addons.

Esta línea sirve para que el ERP reconozca la carpeta donde se guardan los módulos a poder descargar



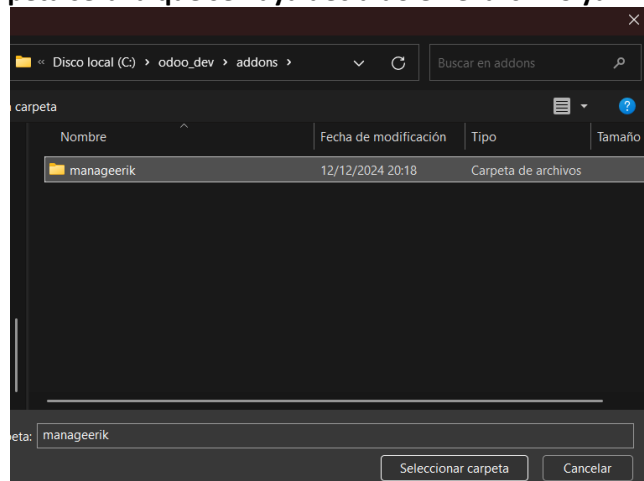
Creación del Módulo:

Para crear el módulo, habrá que ir a Docker y acceder al contenedor llamado "Odoo". Una vez en este, habrá que ir a la sección "Exec" y ejecutar los siguientes comandos:

cd sirve para facilitar la creación, ya que se accede a la carpeta de addons de forma directa y no hace falta usar rutas absolutas en el comando scaffold

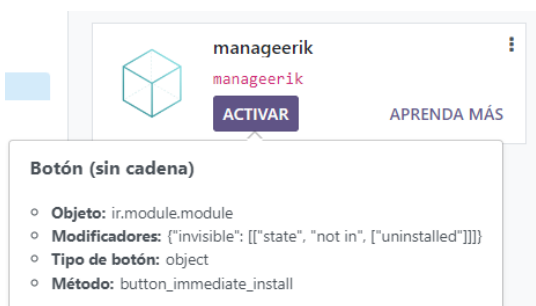
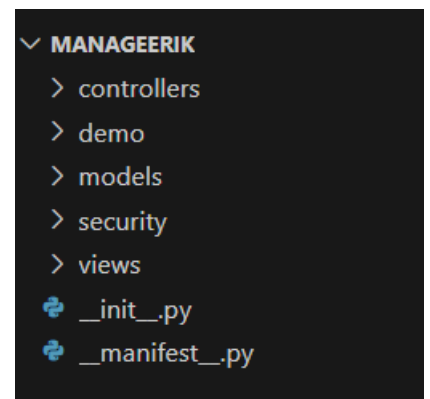
Odoo Scaffold permite crear el módulo de Odoo con toda su estructura de ficheros, facilitando bastante el trabajo.

Lo único que queda es acceder desde el IDE deseado al directorio donde se guarda el módulo en la máquina física, **esa carpeta será la que se haya decidido en el archivo yaml**



La estructura de ficheros será la siguiente:

- **Controllers** (Python): Contiene clases con métodos para acceder a la información de los modelos (por ejemplo, utilizando una API)
- **Models** (Python): Son las distintas tablas del módulo
- **Security** (CSV): Guarda el acceso a los distintos modelos del módulo para los usuarios y grupos de usuarios
- **Views** (XML): Contiene las vistas del módulo, todo lo relacionado a la interfaz gráfica
- **__init__.py**: Carga las carpetas que puede abrir Odoo
- **__manifest__.py**: Carga toda la información del módulo: Las vistas a abrir, dependencias, datos del módulo, ubicación de los datos de demostración...



Lo único que queda es iniciar el módulo en el ERP creado. Esto se hará buscando el nombre del módulo (Quitando la opción de "Aplicación" del buscador de aplicaciones de Odoo) y dando a activar una vez aparezca el deseado

2.2 Modelo Relacional

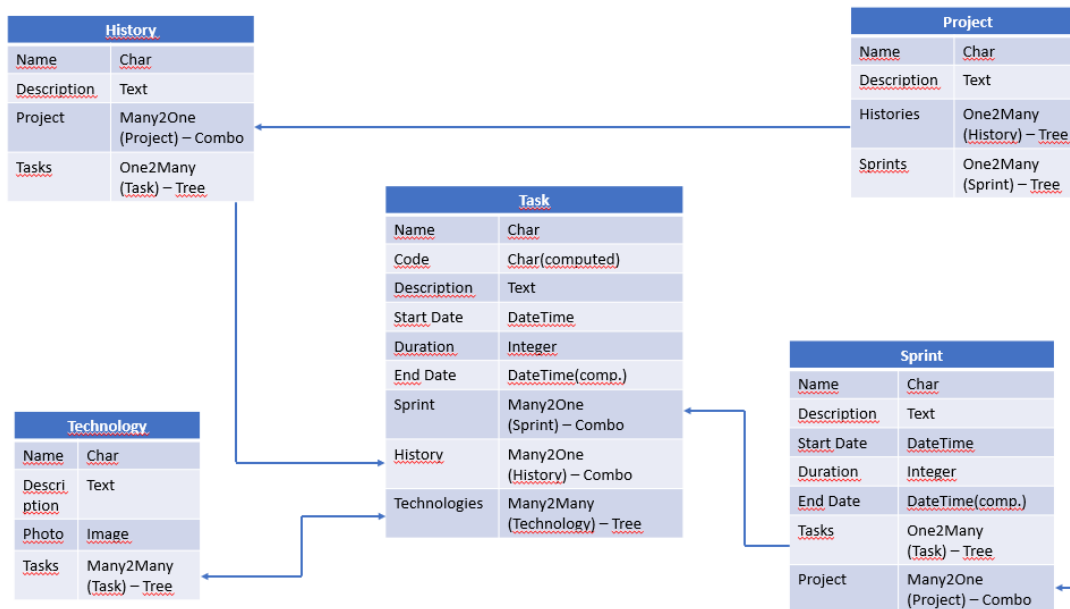
El enunciado del proyecto nos pedía introducir una serie de tablas relacionadas entre ellas, siendo las tablas las siguientes:

- **Historia de Usuario:** Con un nombre y una descripción de esta
- **Proyecto:** Con un nombre y una descripción
- **Tarea:** Con un nombre, un código computado, una descripción, una duración y fechas de inicio y fin
- **Tecnología del SCRUM:** Con un nombre y una descripción
- **Sprint:** Con un nombre, una descripción, una duración y unas fechas de inicio y fin

A mayores de esas tablas, teníamos las **relaciones** entre ellas:

- Una historia puede pertenecer a un único proyecto, y un proyecto puede tener varias historias de usuario
- Una historia de usuario puede tener varias tareas, mientras que cada tarea aparece en una única historia
- Cada tarea puede usar varias tecnologías, y una tecnología puede aparecer en varias tareas
- Los sprints pueden tener varias tareas, mientras que cada tarea solo puede aparecer en un sprint
- Un proyecto puede tener varios sprints, mientras que un sprint solo puede pertenecer a un proyecto.

En su consecuencia, el modelo relacional del proyecto propuesto es el siguiente:



2.3 Procedimiento y Pruebas de funcionamiento

Al tener que hacer tanto el modelo como las vistas del proyecto, me decanté en comenzar por tener todos los modelos y de ahí continuar con la vista, ya que no se consigue nada haciendo los modelos si no ves qué estás haciendo.

2.3.1 Creación de los modelos

Los modelos son la parte principal de la arquitectura MVC de Odoo. Son los objetos de Python que utiliza el Mapeo Objeto-Relación (ORM) del ERP para guardar tablas de datos. Cada clase representa una tabla distinta en la base de datos, tablas que pueden tener distinta cantidad de columnas de varios tipos:

- Char() -> Cadenas de texto sencillas, normalmente no superiores a una línea
- Text() -> Cadenas de texto complejas, con distintos párrafos y líneas
- Integer() -> Número entero
- Float() -> Número decimal con coma flotante
- Boolean() -> Expresión booleana que deja elegir entre verdadero y falso
- Selection() -> Abre un menú, otorgando al usuario elegir un valor en una selección
- Bytes() -> Permite guardar archivos en formato binario
- Datetime() -> Guarda los datos en formato de fecha y hora

El proyecto comenzará sin ningún tipo de relación ni campo computado para evitar problemas al principio. Todos los módulos comienzan con 3 atributos principales, 2 de ellos obligatorios por Odoo:

- `_name` -> Es un campo obligatorio para identificar la tabla en el mapeo de tablas
- `_description` -> Igual que `_name`, este es un campo obligatorio que identifica a la tabla en el mapeo
- `id` -> No es obligatorio para la creación de un modelo, pero no está de menos ponerlo ya que **es un campo auto computado** por el ORM de Odoo que identifica cada dato del modelo

Ejemplo de modelo:

```
# class manageerik(models.Model):
#     _name = 'manageerik.manageerik'
#     _description = 'manageerik.manageerik'

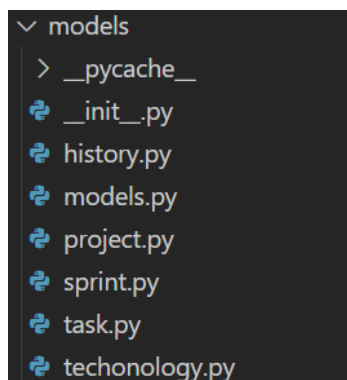
#     id = fields.Integer()
```

Para facilitar la lectura del proyecto, voy a dividir cada módulo de Odoo en un fichero de Python diferente, cada uno de ellos con un modelo del ORM.

Si se sigue esta práctica, es necesario importar los siguientes valores en el archivo `__init__.py` del directorio:

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

from . import models
from . import project
from . import task
from . import history
from . import sprint
from . import technology
```



2.3.1.1 Tabla 'Project'

Siguiendo la tabla del modelo sacado anteriormente y, quitando las relaciones de forma temporal, nos quedarían los bloques de Nombre y Descripción. Nombre se guardará con una cadena de caracteres simple, ya que no deja de ser un nombre corto, mientras que Descripción se guardará como un texto de formato más amplio por si necesita más espacio de lo que puede ofrecer una cadena simple. El nombre será obligatorio, al contrario que la descripción.

Project	
<u>Name</u>	<u>Char</u>
<u>Description</u>	Text
<u>Histories</u>	One2Many (History) – <u>Tree</u>
<u>Sprints</u>	One2Many (Sprint) – <u>Tree</u>

Resultado:

```
from odoo import models, fields, api

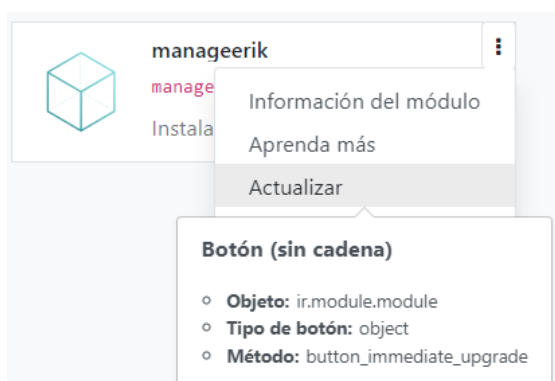
class project(models.Model):
    _name = 'manageerik.project'
    _description = 'manageerik.project'

    id = fields.Integer()
    name = fields.Char(required=True)
    description = fields.Text()
```

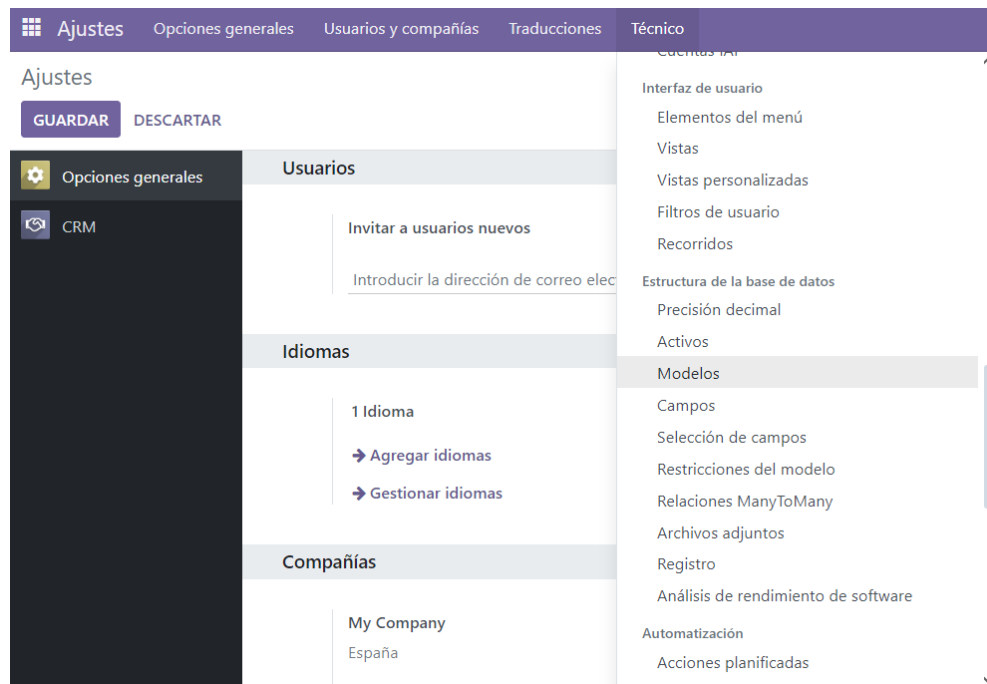
Comprobación:

Para hacer todas las comprobaciones cada vez que se cree un nuevo modelo, se deberá hacer lo siguiente:

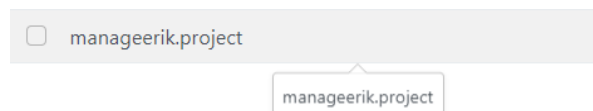
1. Reiniciar el contenedor de Odoo de Docker
2. Actualizar el módulo en la sección Aplicaciones del ERP



Ahora, para ver los modelos creados, habrá que ir a **Ajustes -> Técnico -> Modelos**.



Una vez dentro, se busca el nombre del módulo (atributo `_name` de la clase del módulo), y se selecciona para ver su información



Se podrá ver que, efectivamente, se han creado los campos introducidos (junto algunos otros que crea y gestiona Odoo)

Nombre de campo	Etiqueta de campo	Tipo de campo
<code>_last_update</code>	Last Modified on	fecha y hora
<code>create_date</code>	Created on	fecha y hora
<code>create_uid</code>	Created by	many2one
<code>description</code>	Description	texto
<code>display_name</code>	Display Name	carácter
<code>id</code>	ID	entero
<code>name</code>	Name	carácter
<code>write_date</code>	Last Updated on	fecha y hora
<code>write_uid</code>	Last Updated by	many2one

2.3.1.2 Tabla ‘Task’

La tabla Task, quitando de momento las relaciones y los computados, tendría los campos de nombre, descripción, código, fecha de inicio, fecha de fin y duración. Nombre y descripción serán igual que en historia, mientras que el código, las fechas serán de tipo DateTime para guardar fecha y hora de comienzo y fin, además de la duración del evento. Los campos Code y EndDate aún no estarán computados, por lo que, de momento simplemente estarán inactivos (readonly=True)

Task	
Name	Char
Code	Char(computed)
Description	Text
Start Date	DateTime
Duration	Integer
End Date	DateTime(comp.)
Sprint	Many2One (Sprint) – Combo
History	Many2One (History) – Combo
Technologies	Many2Many (Technology) – Tree

Resultado:

```
from odoo import models, fields, api

class task(models.Model):
    _name = 'manageerik.task'
    _description = 'manageerik.task'

    id = fields.Integer()
    name = fields.Char(required=True)
    code = fields.Char(readonly=True)
    description = fields.Text()
    start_date = fields.Datetime(required=True),
    duration = fields.Integer(),
    end_date = fields.Datetime(readonly=True)
```

Comprobación:

☐ manageerik.task

manageerik.task

Nombre de campo	Etiqueta de campo	Tipo de campo
__last_update	Last Modified on	fecha y hora
code	Code	carácter
create_date	Created on	fecha y hora
create_uid	Created by	many2one
description	Description	texto
display_name	Display Name	carácter
end_date	End Date	fecha y hora
id	ID	entero
name	Name	carácter
write_date	Last Updated on	fecha y hora
write_uid	Last Updated by	many2one

2.3.1.3 Tabla ‘Sprint’

La tabla Sprint sin relaciones tiene los datos de nombre, descripción, duración y fechas de inicio y fin. Funcionan igual que en la anterior tabla

Resultado:

Sprint	
Name	Char
Description	Text
Start Date	DateTime
Duration	Integer
End Date	DateTime(comp.)
Tasks	One2Many (Task) – Tree
Project	Many2One (Project) – Combo

```
from odoo import models, fields, api
```

```
class sprint(models.Model):
    _name = 'manageerik.sprint'
    _description = 'manageerik.sprint'

    id = fields.Integer()
    name = fields.Char(required=True)
    description = fields.Text()
    start_date = fields.Datetime(required=True),
    duration = fields.Integer(),
    end_date = fields.Datetime(readonly=True)
```

Comprobación:

<input type="checkbox"/>	manageerik.sprint	manageerik.sprint
Nombre de campo	Etiqueta de campo	Tipo de campo
_last_update	Last Modified on	fecha y hora
create_date	Created on	fecha y hora
create_uid	Created by	many2one
description	Description	texto
display_name	Display Name	carácter
end_date	End Date	fecha y hora
id	ID	entero
name	Name	carácter
write_date	Last Updated on	fecha y hora
write_uid	Last Updated by	many2one

2.3.1.4 Tabla ‘Technology’

La tabla de tecnologías tiene el nombre y descripción al igual que el resto de tablas, pero tiene un valor a mayores: **La foto**, un valor que se guarda en formato Image (Una extensión del formato Bytes que sirve únicamente para guardar imágenes)

Technology	
Name	Char
Description	Text
Photo	Image
Tasks	Many2Many (Task) – Tree

Resultado:

```
from odoo import models, fields, api

class techonology(models.Model):
    _name = 'manageerik.technology'
    _description = 'manageerik.technology'

    id = fields.Integer()
    name = fields.Char(required=True)
    description = fields.Text()
    image = fields.Image()
```

Comprobación:

<input type="checkbox"/> manageerik.technology	manageerik.technology	
Nombre de campo	Etiqueta de campo	Tipo de campo
_last_update	Last Modified on	fecha y hora
create_date	Created on	fecha y hora
create_uid	Created by	many2one
description	Description	texto
display_name	Display Name	carácter
id	ID	entero
image	Image	binario
name	Name	carácter
write_date	Last Updated on	fecha y hora
write_uid	Last Updated by	many2one

2.3.1.5 Tabla ‘History’

La tabla de historias sin relaciones solo tiene dos valores: el nombre y la descripción, comunes en todas las clases previas

Resultado:

History	
Name	Char
Description	Text
Project	Many2One (Project) – Combo
Tasks	One2Many (Task) – Tree

```
from odoo import models, fields, api

class history(models.Model):
    _name = 'manageerik.history'
    _description = 'manageerik.history'

    id = fields.Integer()
    name = fields.Char(required=True)
    description = fields.Text()
```

Comprobación:

<input type="checkbox"/> manageerik.history	manageerik.history	
Nombre de campo	Etiqueta de campo	Tipo de campo
__last_update	Last Modified on	fecha y hora
create_date	Created on	fecha y hora
create_uid	Created by	many2one
description	Description	texto
display_name	Display Name	carácter
id	ID	entero
name	Name	carácter
write_date	Last Updated on	fecha y hora
write_uid	Last Updated by	many2one

2.3.2 Dar acceso a la información

Una vez creados los módulos, para poder acceder a todas las futuras vistas que se vayan a crear, hace falta poner permisos a los usuarios desde el archivo **ir.model.access.csv**, en el siguiente formato:

```
id,name,model_id:id,group_id:id,perm_read,perm_write,perm_create,perm_unlink
access_manageerik_history,manageerik.history,model_manageerik_history,base.group_user,1,1,1,1
access_manageerik_project,manageerik.project,model_manageerik_project,base.group_user,1,1,1,1
access_manageerik_sprint,manageerik.sprint,model_manageerik_sprint,base.group_user,1,1,1,1
access_manageerik_task,manageerik.task,model_manageerik_task,base.group_user,1,1,1,1
access_manageerik_technology,manageerik.technology,model_manageerik_technology,base.group_user,1,1,1,1
```

2.3.3 Creación de las vistas

Las vistas son el siguiente punto a seguir en este modelo, son la parte gráfica de la arquitectura MVC de Odoo, donde el usuario puede visualizar los datos del modelo e interactuar con ellos. Están creados en ficheros XML con las distintas formas de visualizar los datos. A lo largo de las distintas vistas, habrá dos formas de visualizar los datos: En forma de árbol (o tree) y como formulario. Siguen una misma estructura, cambiando un par de valores:

```
<record model="ir.ui.view" id="id_vista">
  <field name="name">nombre_vista</field>
  <field name="model">modulo.modelo</field>
  <field name="arch" type="xml">
    <form string="NombreFormulario " >
      <sheet>
        <group name="nombreGrupo">
          <field name="nombre"/>
          <...>
        </group>
      </sheet>
    </form>
  </field>
</record>
<record model="ir.ui.view" id=" id_vista ">
  <field name="name">nombre_vista</field>
  <field name="model">modulo.modelo</field>
  <field name="arch" type="xml">
    <tree>
      <field name="nombre"/>
      <...>
    </tree>
  </field>
</record>
```


También hay que crear la **acción**.

Es importante introducir los campos en cada grupo de etiquetas como es conveniente. Además, en el archivo `__manifest__.py` tiene que aparecer las vistas reflejadas en el apartado *data*:

```
# always loaded
'data': [
    'security/ir.model.access.csv',
    'views/views.xml',
    'views/templates.xml',
    'views/history.xml',
    'views/project.xml',
    'views/sprint.xml',
    'views/task.xml',
    'views/technology.xml',
],
```

Las vistas aparecerán una vez se actualice el módulo y siguiendo los siguientes pasos:

1. Abrir la vista: Esto se hace yendo al menú debug que aparece siempre en la parte superior una vez has activado el modo desarrollador en Odoo. Entre las opciones disponibles aparece “Abrir Vista”, el cual te permite buscar una vista y previsualizarla

Seleccionar una vista

Vista [manageerik x] Buscar...

Filtros Agrupar por Favoritos 1-10 / 10 < >

Nombre de la vista	Tipo de vista	Modelo	ID externo	Vista heredada
Project Form	Formulario	manageerik.project	manageerik.manageerik_project_form	
Task Form	Formulario	manageerik.task	manageerik.manageerik_task_form	
Project View	Árbol	manageerik.project	manageerik.manageerik_project_tree	
Task View	Árbol	manageerik.task	manageerik.manageerik_task_tree	
Sprint Form	Formulario	manageerik.sprint	manageerik.manageerik_sprint_form	
Sprint View	Árbol	manageerik.sprint	manageerik.manageerik_sprint_tree	
History Form	Formulario	manageerik.history	manageerik.manageerik_history_form	
History View	Árbol	manageerik.history	manageerik.manageerik_history_tree	
Technology Form	Formulario	manageerik.technology	manageerik.manageerik_technology_form	
Technology View	Árbol	manageerik.technology	manageerik.manageerik_technology_tree	

NUEVO CERRAR

Ejecutar pruebas JS

Ejecutar pruebas JS Mobile

Ejecutar prueba de clic en todos lados

Abrir vista

Desactivar recorridos

Comenzar recorrido

Editar Acción

Ver campos

Gestionar filtros

Ver permisos de acceso

Ver reglas de registro

Establecer valores por defecto

Obtener vista

Editar vista:Form

Activar depuración de activos

Activar pruebas para depuración de activos

Regenerar paquetes de activos

Convertirse en superusuario

Abandonar las herramientas de desarrollador

☐ Habilitar análisis de rendimiento de software

Lo único necesario tras esto es pulsar en la vista que se desee abrir y se iniciará la vista en primer plano.

2.3.3.1 Vista 'Project'

Con los datos de la vista de proyectos, acabaría siendo así:

```
<odoo>
  <data>
    <record model="ir.ui.view" id="manageerik_project_form">
      <field name="name">Project Form</field>
      <field name="model">manageerik.project</field>
      <field name="arch" type="xml">
        <form string="Project Form">
          <sheet>
            <group name="groupTop">
              <field name="name"/>
              <field name="description"/>
            </group>
          </sheet>
        </form>
      </field>
    </record>
    <record model="ir.ui.view" id="manageerik_project_tree">
      <field name="name">Project View</field>
      <field name="model">manageerik.project</field>
      <field name="arch" type="xml">
        <tree>
          <field name="name"/>
          <field name="description"/>
        </tree>
      </field>
    </record>
  </data>
</odoo>
```

Project	
Name	Char
Description	Text
Histories	One2Many (History) – Tree
Sprints	One2Many (Sprint) – Tree

Seleccionar una vista

Vista manage + Buscar...

▼ Filtros Agrupar por ★ Favoritos 1-2 / 2 < >

Nombre de la vista	Tipo de vista	Modelo	ID externo	Vista heredada
Project Form	Formulario	manageerik.project	manageerik.manageerik_project_form	
Project View	Árbol	manageerik.project	manageerik.manageerik_project_tree	

Aplicaciones / Nuevo

Acción Nuevo

Name ?

Description ?

Aplicaciones / Project View

Buscar...

NUEVO

▼ Filtros Agrupar por ★ Favoritos

<input type="checkbox"/> Name	Description


2.3.3.2 Vista 'Task'

En esta se pueden apreciar los campos que no permiten ser editados a mano por el usuario, ya que están en un tono más grisáceo.

```
<record model="ir.ui.view" id="manageerik_project_form">
  <field name="name">Task Form</field>
  <field name="model">manageerik.task</field>
  <field name="arch" type="xml">
    <form string="Task Form">
      <sheet>
        <group name="groupTop">
          <group name="groupLeft">
            <field name="name" />
            <field name="code" />
            <field name="description" />
          </group>
          <group name="groupRight">
            <field name="start_date" />
            <field name="end_date" />
            <field name="duration" />
          </group>
        </group>
      </sheet>
    </form>
  </field>
</record>
<record model="ir.ui.view" id="manageerik_task_tree">
  <field name="name">Task View</field>
  <field name="model">manageerik.task</field>
  <field name="arch" type="xml">
    <tree>
      <field name="name" />
      <field name="code" />
      <field name="description" />
      <field name="start_date" />
      <field name="end_date" />
      <field name="duration" />
    </tree>
  </field>
</record>
```

Task	
Name	Char
Code	Char(computed)
Description	Text
Start Date	DateTime
Duration	Integer
End Date	DateTime(comp.)
Sprint	Many2One (Sprint) – Combo
History	Many2One (History) – Combo
Technologies	Many2Many (Technology) – Tree

Nombre de la vista	Tipo de vista	Modelo	ID externo
Task Form	Formulario	manageerik.task	manageerik.manageerik_task_form
Task View	Árbol	manageerik.task	manageerik.manageerik_task_tree



Ajustes / Nuevo 

Name ?
Code ?
Description ?

Start Date ?
End Date ?
Duration ? 0

Ajustes / Task View

Buscar...

▼ Filtros  Agrupar por  Favoritos

<input type="checkbox"/> Name	Code	Description	Start Date	End Date	Du...
-------------------------------	------	-------------	------------	----------	-------

2.3.3.3 Vista 'Sprint'

```
<record model="ir.ui.view" id="manageerik_sprint_form">
  <field name="name">Sprint Form</field>
  <field name="model">manageerik.sprint</field>
  <field name="arch" type="xml">
    <form string="sprint Form">
      <sheet>
        <group name="groupTop">
          <group name="groupLeft">
            <field name="name" />
            <field name="description" />
          </group>
          <group name="groupRight">
            <field name="start_date" />
            <field name="end_date" />
            <field name="duration" />
          </group>
        </group>
      </sheet>
    </form>
  </field>
</record>
<record model="ir.ui.view" id="manageerik_sprint_tree">
  <field name="name">Sprint View</field>
  <field name="model">manageerik.print</field>
  <field name="arch" type="xml">
    <tree>
      <field name="name" />
      <field name="code" />
      <field name="start_date" />
      <field name="end_date" />
      <field name="duration" />
    </tree>
  </field>
</record>
```

Sprint	
Name	Char
Description	Text
Start Date	DateTime
Duration	Integer
End Date	DateTime(comp.)
Tasks	One2Many (Task) – Tree
Project	Many2One (Project) – Combo

Nombre de la vista	Tipo de vista	Modelo	ID externo	Vista heredada
⚙ Sprint Form	Formulario	manageerik.sprint	manageerik.manageerik_sprint_form	
⚙ Sprint View	Árbol	manageerik.sprint	manageerik.manageerik_sprint_tree	

Ajustes / Sprint View

Buscar...

NUEVO

Filtros Agrupar por Favoritos

<input type="checkbox"/> Name	Description	Start Date	End Date	Da...

Ajustes / Nuevo

Acción Nuevo

Name ?	Start Date ?
Description ?	End Date ?
	Duration ? 0

2.3.3.4 Vista 'Technology'

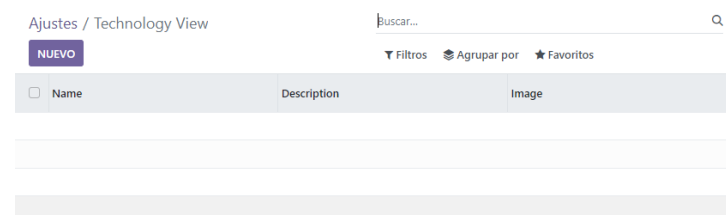
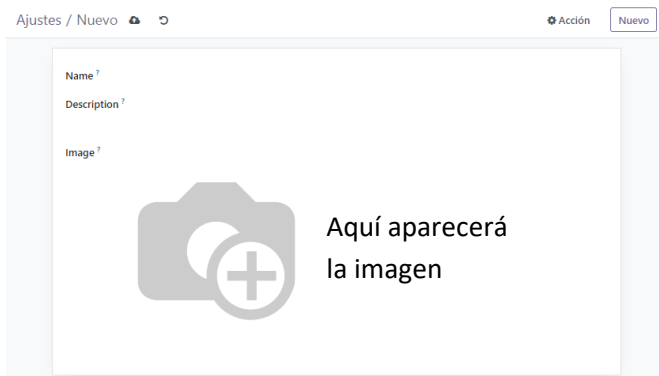
Esta vista trae una diferencia con las demás, ya que tiene un campo de imagen. Para poder ver la imagen correctamente en la vista, hay que añadir el atributo `widget="image"` al campo de la imagen

Technology	
Name	Char
Description	Text
Photo	Image
Tasks	Many2Many (Task) - Tree

```
<record model="ir.ui.view" id="manageerik_technology_form">
  <field name="name">Technology Form</field>
  <field name="model">manageerik.technology</field>
  <field name="arch" type="xml">
    <form string="Technology Form">
      <sheet>
        <group name="groupTop">
          <field name="name" />
          <field name="description" />
          <field name="image" widget="image"/>
        </group>
      </sheet>
    </form>
  </field>
</record>
<record model="ir.ui.view" id="manageerik_technology_tree">
  <field name="name">Technology View</field>
  <field name="model">manageerik.technology</field>
  <field name="arch" type="xml">
    <tree>
      <field name="name" />
      <field name="description" />
      <field name="image" widget="image"/>
    </tree>
  </field>
</record>
```

Comprobación:

Nombre de la vista	Tipo de vista	Modelo	ID externo	Vista heredada
Technology View	Árbol	manageerik.technology	manageerik.manageerik_technology_tree	
Technology Form	Formulario	manageerik.technology	manageerik.manageerik_technology_form	



2.3.3.5 Vista 'History'

```
<record model="ir.ui.view" id="manageerik_history_form">
  <field name="name">History Form</field>
  <field name="model">manageerik.history</field>
  <field name="arch" type="xml">
    <form string="History Form">
      <sheet>
        <group name="groupTop">
          <field name="name"/>
          <field name="description"/>
        </group>
      </sheet>
    </form>
  </field>
</record>
<record model="ir.ui.view" id="manageerik_history_tree">
  <field name="name">History View</field>
  <field name="model">manageerik.history</field>
  <field name="arch" type="xml">
    <tree>
      <field name="name"/>
      <field name="description"/>
    </tree>
  </field>
</record>
```

Comprobación:

Nombre de la vista	Tipo de vista	Modelo	ID externo	Vista heredada
History View	Árbol	manageerik.history	manageerik.manageerik_history_tree	
History Form	Formulario	manageerik.history	manageerik.manageerik_history_form	

Ajustes / History View

Buscar...

NUEVO

Filtros
 Agrupar por
 Favoritos

<input type="checkbox"/> Name	Description

Name ?

Description ?

History	
Name	Char
Description	Text
Project	Many2One (Project) – Combo
Tasks	One2Many (Task) – Tree

2.3.4 Creación de las acciones

Las acciones sirven para definir distintos comportamientos del ERP de Odoo, como pueden ser los inicios de sesión, las acciones de dar a un botón, o la carga de vistas, que es para lo que se va a usar en este caso.

ir.actions.act_window permite especificar los tipos de vista que van a cargar de un modelo. Permite también especificar un dominio de objetos, utilizado para segregar conjuntos de datos y mostrar solo uno de ellos. El prototipo de modelo de acciones será el siguiente:

```
<record model="ir.actions.act_window" id="action_manageerik_technology_form">
  <field name="name">Technology List</field>
  <field name="type">ir.actions.act_window</field>
  <field name="res_model">manageerik.technology</field>
  <field name="view_mode">tree,form</field>
  <field name="help" type="html">
    <p class="oe_view_nocontent_create">
      Technology
    </p>
    <p> Click <strong> 'Create' </strong> to add new technologies </p>
  </field>
</record>
```

2.3.5 Orden de los menús

Para acceder a un módulo es necesario especificar una serie de botones de menú. Estos menús tendrán una jerarquía de hasta tres niveles que definirá la forma que tendrán de aparecer en la interfaz.

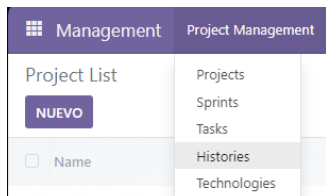
Los menús tienen 4 atributos:

- name: Nombre que se muestra en ese menú o submenú
- id: Identificador del menú
- parent (opcional): Menú padre de este. Si no tiene parent automáticamente es un menú absoluto (aparece en la sección de aplicaciones)
- action: La acción que se activa cuando se pulsa en ese menú

Para cargarlos de una forma óptima, los menús deben estar declarados en la parte inferior del último fichero que carga el manifiesto del módulo (en mi caso, Technologies, pg.17)

```
<menuitem name="Management" id="menu_manageerik_root"/>
<menuitem name="Project Management" id="menu_manageerik_manage_list" parent="menu_manageerik_root"/>
<menuitem name="Projects" id="menu_manageerik_project" parent="menu_manageerik_manage_list" action="action_manageerik_project_form"/>
<menuitem name="Sprints" id="menu_manageerik_sprint" parent="menu_manageerik_manage_list" action="action_manageerik_sprint_form"/>
<menuitem name="Tasks" id="menu_manageerik_task" parent="menu_manageerik_manage_list" action="action_manageerik_task_form"/>
<menuitem name="Histories" id="menu_manageerik_history" parent="menu_manageerik_manage_list" action="action_manageerik_history_form"/>
<menuitem name="Technologies" id="menu_manageerik_technology" parent="menu_manageerik_manage_list"
action="action_manageerik_technology_form"/>
```

Resultado:



2.4 Ampliaciones

Aunque este proyecto tiene bastante contenido, hace falta realizar una serie de ampliaciones para ofrecer una mejor experiencia al usuario y mejorar su funcionalidad.

2.4.1 Relaciones

Al principio se marcaron una serie de relaciones que debía cumplir el proyecto, y por ahora no se han cumplido en absoluto, por lo que hay que crearlas.

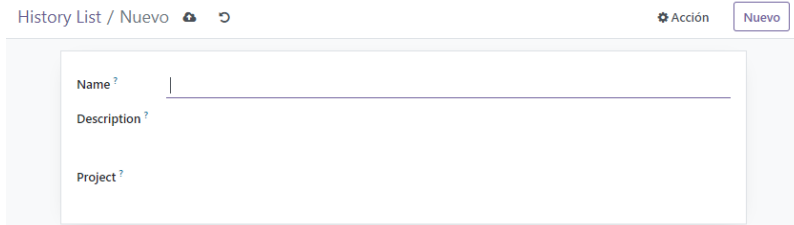
Hay 3 tipos de relación: One2Many (Un objeto solo puede tener una ocurrencia de otro), Many2One (Un objeto puede tener varios otros) y Many2Many (Muchos objetos aparecen en muchos otros)

2.4.1.1 Relación Historia-Proyecto (1:N)

"Una historia puede pertenecer a un único proyecto, y un proyecto puede tener varias historias de usuario": Va a ser una relación 1:N, donde el campo Many2One pertenece al proyecto y el One2Many a la historia.

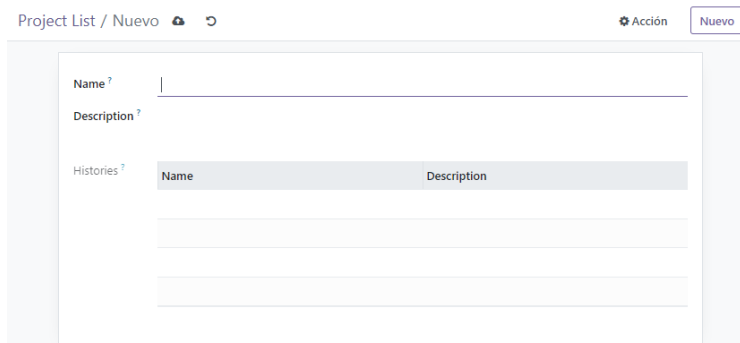
En History:

```
project = fields.Many2one("manageerik.project", ondelete = "cascade",
string="Project")
```



En Project:

```
histories = fields.One2many(string="Histories",
comodel_name="manageerik.history", inverse_name="project", readonly=True)
```





2.4.1.2 Relación Historia-Tarea (1:N)

"Una historia de usuario puede tener varias tareas, mientras que cada tarea aparece en una única historia": En este caso, las historias van a ser el campo Many2One y las tareas serán One2Many

En History:

```
tasks = fields.One2many(string = "Tasks", comodel_name="manageerik.task",
inverse_name="history", readonly=True)
```

History List / Nuevo   Acción Nuevo

Name ?

Description ?



Project ?

Tasks ?

N...	C...	D...	Start Date	End Date	Dura...	Hi...
<input type="text"/>						
<input type="text"/>						

En Task:

```
history = fields.Many2one("manageerik.history", required = True, ondelete =
"cascade", string="History")
```

Task List / Nuevo   Acción Nuevo

Name ?

Code ?

Description ?

Start Date ?

End Date ?

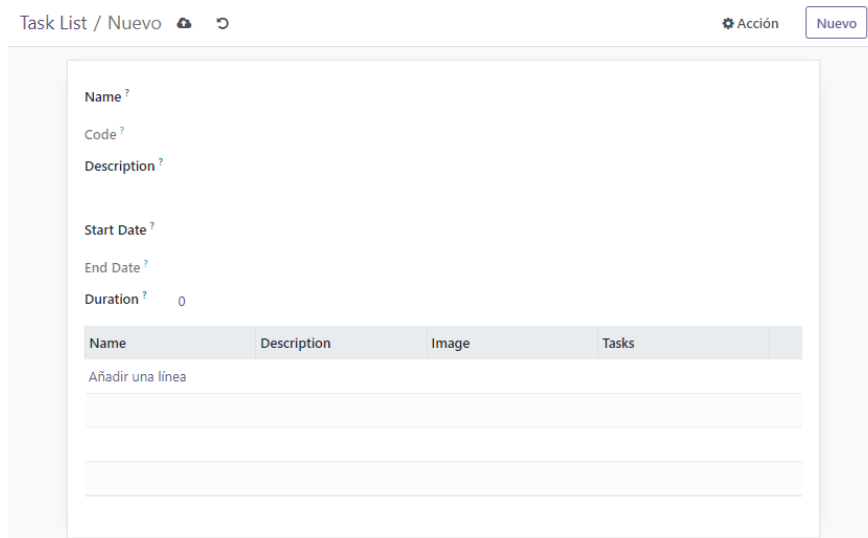
Duration ?

2.4.1.3 Tarea-Tecnología (N:N)

"Cada tarea puede usar varias tecnologías, y una tecnología puede aparecer en varias tareas": Esta es una relación Many2Many, donde una tarea puede usar varias tecnologías y una misma tecnología puede aparecer en varias tareas

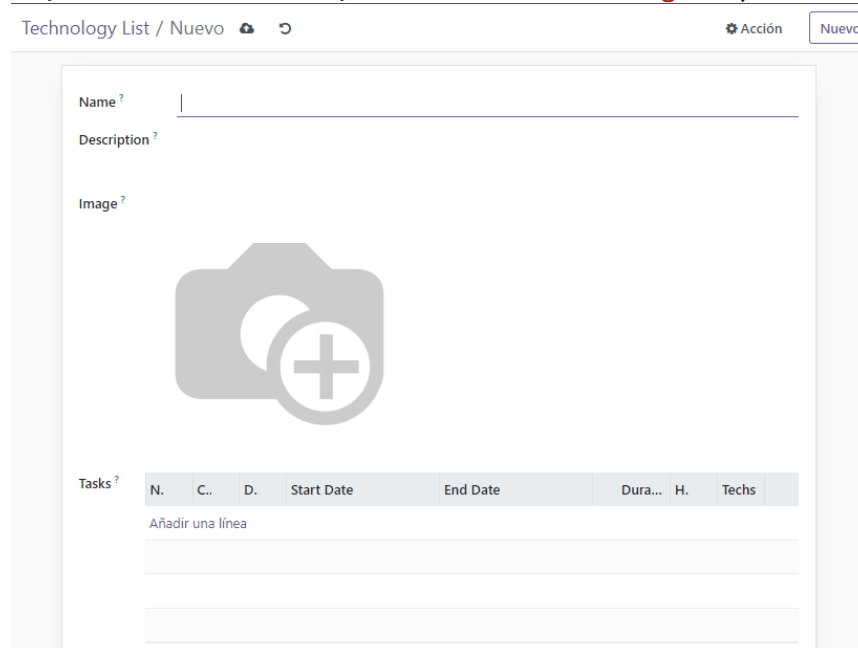
En Task:

```
technologies = fields.Many2many("manageerik.technology", string="Techs",
relation = "techs_tasks", column1 = "technologies", column2 = "tasks")
```



En Technology:

```
tasks = fields.Many2many("manageerik.task", string="Tasks", relation =
"techs_tasks", column1 = "tasks", column2 = "technologies")
```





2.4.1.4 Sprint-Tarea (1:N)

"Los sprints pueden tener varias tareas, mientras que cada tarea solo puede aparecer en un sprint":
Esta es una relación 1:N donde las tareas son el campo One2Many y los sprints son el campo Many2One

Sprint:

```
tasks = fields.One2many(string="tasks", comodel_name="manageerik.task",
inverse_name="sprint", readonly=True)
```

Sprint List / Nuevo   Acción Nuevo

Name ?

Description ?

Start Date ?



End Date ?

Duration ? 0

tasks ?

N	C	D	Start Date	End Date	Dura...	H	Tec...	S

Task:

Task List / Nuevo   Acción Nuevo

Name ?

Code ?

Description ?

History ?

Start Date ?

End Date ?

Duration ? 0

Techs ?

Name	Description	Image	Tasks
Añadir una línea			




Sprint ?

2.4.1.5 Proyecto-Sprint(1:N)

"Un proyecto puede tener varios sprints, mientras que un sprint solo puede pertenecer a un proyecto": Es una relación 1:N, donde los proyectos son el campo Many2One y los sprints son el campo One2Many

Sprint:

```
project = fields.Many2one("manageerik.project", string="project", required = False, ondelete = "cascade")
```

Sprint List / Nuevo    Acción Nuevo

Name ?

Description ?

project ?

Start Date ?

End Date ?




Duration ? 0

tasks ?

N	C	D	Start Date	End Date	Dura... H	Tec... S

Project:

```
sprints = fields.One2many(string="Sprints", comodel_name="manageerik.sprint", inverse_name="project", readonly=True)
```

Project List / Nuevo    Acción Nuevo

Name ?

Description ?

Histories ?

Name	Description	Project	Tasks

Sprints ?

N...	D...	Start Date	End Date	Dura...	tasks	pr...

2.4.2 Computación de campos

Hay veces que no interesa que el usuario se vea obligado a escribir en todos los campos o se le facilite distintas operaciones a la hora de crear datos, por lo que se usan los **campos computados**: Son campos ReadOnly que solo permiten su edición por medio de funciones definidas en la clase. Un ejemplo es el atributo "Id" que ofrece Odoo: El usuario no lo puede cambiar, pero cada vez que se crea un nuevo dato en una tabla, ese Id se incrementa automáticamente.

2.4.2.1 Computación del código de la tarea

El atributo "code" de las tareas estaba previamente marcado como readonly hasta llegar a este punto.

Lo que va a hacer esta función es *coger el nombre del dato y añadir TSK_ al inicio. Si el dato no tuviera nombre, el campo se rellenaría como TSK_noCode*

```
code = fields.Char(compute="_get_code", store = False)
```

```
@api.depends('name')
def _get_code(self):
    for task in self:
        if task.name == False:
            task.code = "TSK_noCode"
        else:
            task.code = f"TSK_{task.name}"
```

Comprobación:

Name ?		Name ?	Prueba
Code ?	TSK_noCode	Code ?	TSK_Prueba

2.4.2.2 Computación de las fechas de fin

Otro campo previamente bloqueado era el de la fecha de fin, tanto en sprints como en tasks.

Ahora, este campo va a ser editado de la siguiente forma:

Si la fecha de inicio es de tipo datetime (tipo fecha) y la duración es mayor a 0, la fecha final será la fecha de inicio más la duración en días.

```
end_date = fields.Datetime(compute="_get_end_date", store=True)
```

```
@api.depends('start_date', 'duration')
def _get_end_date(self):
    for sprint in self:
        if isinstance(sprint.start_date, datetime.datetime) and
sprint.duration > 0:
            sprint.end_date = sprint.start_date +
datetime.timedelta(days=sprint.duration)
        else:
            sprint.end_date = sprint.start_date
```

Start Date ? 23/12/2024 06:26:16
End Date ? 28/12/2024 06:26:16
Duration ? 5

2.4.2.3 Computación del campo Sprint en Tarea

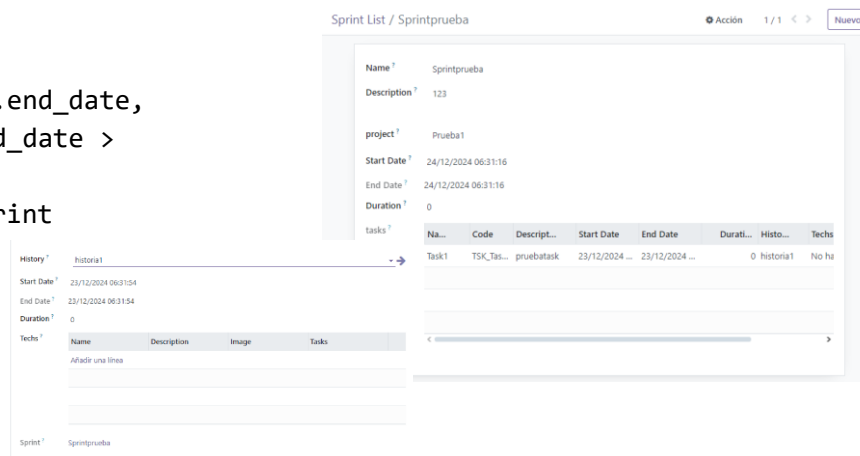
Otro ejemplo de computación es el siguiente donde se computa un campo de relación:

Si se asigna una historia a una tarea, se recorren los sprints de la historia.

Si algún sprint no ha finalizado (su fecha de fin es más alto que la fecha actual) se selecciona como sprint de la tarea.

```
sprint = fields.Many2one("managerik.sprint", ondelete = "cascade",
string="Sprint", compute="_get_sprint")
```

```
@api.depends('history')
def _get_sprint(self):
    for task in self:
        sprints = self.env['managerik.sprint'].search([('project.id', '=',
task.history.project.id)])
        found = False
        for sprint in sprints:
            if isinstance(sprint.end_date,
datetime.datetime) and sprint.end_date >
datetime.datetime.now():
                task.sprint = sprint
                found = True
        if not found:
            task.sprint = False
```



Sprint List / Sprintprueba

Acción 1/1 < > Nuevo

Name ? Sprintprueba
Description ? 123
project ? Prueba1
Start Date ? 24/12/2024 06:31:16
End Date ? 24/12/2024 06:31:16
Duration ? 0

Na...	Code	Descript...	Start Date	End Date	Durati...	Histo...	Techi
Task1	TSK_Tas...	pruebatask	23/12/2024 ...	23/12/2024 ...	0	historia1	No ha

History ? historia1
Start Date ? 23/12/2024 06:31:54
End Date ? 23/12/2024 06:31:54
Duration ? 0

Techi ?

Name	Description	Image	Tasks
Añadir una línea			

Sprint ? Sprintprueba

2.4.2.4 Computación del campo Technologies en History



Este campo computado liga las tecnologías con las historias por medio de una relación Many2Many que solo se ve por el lado de las historias:

Se recorren las tareas que guarda cada historia. Si tienen alguna tecnología, se añaden a la lista de tecnologías de la historia

```
used_technologies = fields.Many2many("manageerik.technology",
compute="_get_used_technologies")

def _get_used_technologies(self):
    for history in self:
        technologies = None
        for task in history.tasks:
            if not technologies:
                technologies = task.technologies
            else:
                technologies = technologies + task.technologies
        history.used_technologies = technologies
```

History List / historia1 Acción 1 / 1 < > Nuevo

Used Technologies ?		Name	Description	Image	Tasks
Tec1	Tecnologia1				1 registro
					
Tec2	Tec2				1 registro
					

2.4.2.5 Extra: Computación del número de tareas de un proyecto

Esta es una idea de por mí que en cierto modo puede ser interesante para que el usuario lleve un control más exacto de las tareas que lleva su proyecto:

Se recorren todos los sprints de un proyecto y se cuentan todas las tareas de cada sprint. El número total de todas las tareas de todos los sprints de un proyecto equivale al total de tareas que un proyecto ha llevado a cabo.

```
totaltasks = fields.Integer(string="Total Tasks", compute = "_get_tasks")
```

```
def _get_tasks(self):
    for project in self:
        project.totaltasks = 0
        num = 0
        for sprint in project.sprints:
            for task in sprint.tasks:
                num = num + 1
        project.totaltasks = num
```

<input type="checkbox"/>	Name	Description	Histories	Sprints	Total Tasks
<input type="checkbox"/>	Prueba1	Proyecto Prueba	1 registro	1 registro	2

2.4.3 Extra: Agrupar Tasks por Sprint

Dentro de las vistas, se pueden realizar agrupaciones en función de distintos valores con un tipo de view:


```
project = fields.Many2one(related='history.project', string="Project",
store=True)

<record model="ir.ui.view" id="view_manageerik_task_filters">
  <field name="name">vista_manageerik_task_search</field>
  <field name="model">manageerik.task</field>
  <field name="arch" type="xml">
    <search string="Filter tasks">
      <filter name="groupby_project" string="Project "
        context="{ 'group_by':project}" help="Group by Project" />
    </search>
  </field>
</record>
```

El filtro lo que hace en este caso es hacer grupos en función del Sprint que tienen asignadas las tareas. Puede ser de ayuda para el usuario para poder guiarse mejor por todas las tasks:

Task List

Sprint x

NUEVO 

Filtros Agrupar por Favoritos 1-3 / 3 < >

<input type="checkbox"/>	Name	Code	Description	Start Date	End Date		Techs	Sprint
<div> <div>▼ Sp1 (1)</div> <div> <div>✓ Sprint</div> <div>Añadir grupo personalizado ▼</div> </div> </div>								
<input type="checkbox"/>	Prueba1	TSK_Prueba1		22/12/2024 09:38:59	22/12/2024 09:38:59	0 Historia2	No hay registros	Sp1
						12		
▼ Sp2 (1)								
<input type="checkbox"/>	hfgf	TSK_hfgf		17/12/2024 10:12:07	29/12/2024 10:12:07	12 Historia3	No hay registros	Sp2
						0		
▼ Ninguno (1)								
<input type="checkbox"/>	3	TSK_3	12	25/12/2024 10:12:22	25/12/2024 10:12:22	0 Hist5	No hay registros	

2.4.4 Herencia de desarrolladores con el módulo Contactos

Dentro de todo lo que se puede hacer con los módulos, está la herencia. Una tabla puede heredar campos, datos y funciones de otra. Esto se consigue con el atributo `_inherit` 'modulo.modelo'.

En este caso, se está haciendo una clase de desarrolladores de proyectos SCRUM, que no dejan de ser contactos

```
class developer(models.Model):
    _name = 'res.partner'
    _inherit = 'res.partner'

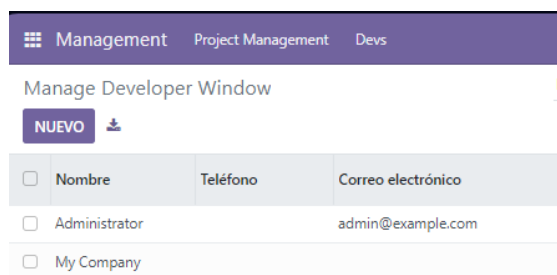
    is_dev = fields.Boolean(default=True, readonly=True)
    technologies = fields.Many2many('manageerik.technology',
relation='developer_technologies', column1='developers',
column2='technologies')
```

Como se puede observar, tiene de nombre y de herencia `res.partner`, que equivale al nombre del modelo de Contactos de Odoo. Desde esta nueva clase se añade la booleana de desarrollador (Siempre va a ser verdadera cuando se cree el contacto desde la ventana de desarrolladores) y una relación con las técnicas que este puede usar.

Luego, en la vista, también puede hacer una herencia de los datos de la tabla principal.

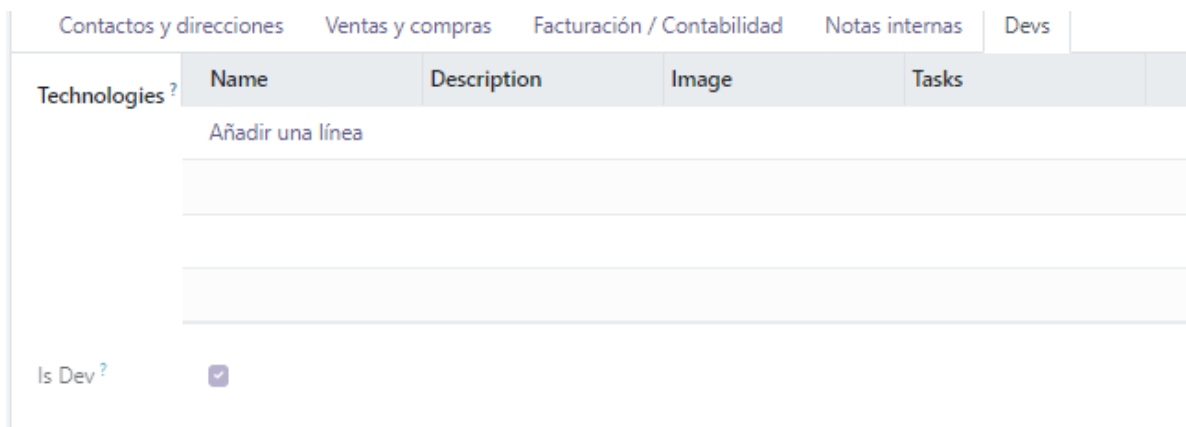
Aparte de eso, también haré una distinción de contactos por medio de un domain para que solo aparezcan los desarrolladores (`is_dev = true`)

```
<record model="ir.actions.act_window" id="manageerik.action_developer_window">
    <field name="name">Manage Developer Window</field>
    <field name="res_model">res.partner</field>
    <field name="view_mode">tree,form</field>
    <field name="domain">[('is_dev', '=', True)]</field>
</record>
<record model="ir.ui.view" id="manageerik.devs_partner_form">
    <field name="name">manage devs form</field>
    <field name="model">res.partner</field>
    <field name="inherit_id" ref="base.view_partner_form"></field>
    <field name="mode">primary</field>
    <field name="arch" type="xml">
    <xpath expr="//sheet/notebook/page[@name='internal_notes']" position="after">
        <page name="devs" string="Devs">
            <group>
                <field name="technologies" />
                <field name="is_dev" />
            </group>
        </page>
    </xpath>
</field>
</record>
```



Nombre	Teléfono	Correo electrónico
<input type="checkbox"/> Administrator		admin@example.com
<input type="checkbox"/> My Company		

La parte de la etiqueta page que hay en la vista de formulario sirve para crear una página nueva en el formulario del contacto. Esta página guardará los datos de tecnologías y la booleana is_dev.



3 Estado del arte

3.1 ERP

Los sistemas ERP son sistemas que integran todos los datos y procesos de una empresa en un sistema unificado. Es un conjunto de programas donde cada uno se encarga de una operación de la empresa relacionada a su producción, gestión de empleados y contactos, ventas...

Se utilizan debido a que se permiten controlar con libertad todos los procesos, solo se instalan los módulos que uno necesite, y un mismo sistema se puede adaptar a distintas empresas

Algunos de los ERPs más conocidos en el mundo empresarial son NetSuite de Oracle, Dynamics 365 de Microsoft y Odoo.

En este trabajo se va a usar Odoo, un ERP conocido por su personalización y la gran cantidad de módulos que ofrece de forma totalmente gratuita, además de un despliegue tanto de forma local como en red.

Para facilitar la creación y manipulación del módulo de Odoo, se va a utilizar la herramienta **Docker**, un gestor de contenedores

Docker funciona por contenedores, una serie de entornos virtuales que empaquetan el código y las dependencias de distintos servicios y aplicaciones y que se ejecutan en el host junto al resto de aplicaciones. Es un método de “virtualización” más rápido y efectivo.

Entrando en el ámbito de Odoo, este servicio se basa en una arquitectura multicapa (Modelo-Vista-Controlador). Tiene 3 capas, la capa de **presentación** (Vista, archivos XML), la capa **lógica** (Controladores, archivos Python que guardan métodos) y la capa de **datos** (Modelos, archivos Python que guardan objetos).

Aparte de esta arquitectura, Odoo destaca por ser **Modular**, es decir: Su estructura se basa en un módulo principal el cual alberga y permite utilizar otros módulos.

3.2 SCRUM

Es una metodología que se utiliza para organizar y gestionar el trabajo en equipo de forma ágil. El método Scrum anima a los equipos a aprender a través de las experiencias y a organizarse mientras trabajan en un problema.

“SCRUM” apareció como concepto en un estudio en 1986 tras ver distintos procesos de desarrollo en productos como coches, ordenadores y fotocopiadoras. Todos estos productos partían de requisitos muy generales y novedosos, por lo que tenían que hacer unos patrones de producción muy similares para todos los productos. Los estudios comparaban esta metodología como los equipos de Rugby, por la colaboración y la forma de disciplina de los equipos (De hecho, el nombre Scrum viene de una formación del equipo, llamada “Melé” en español)

La metodología SCRUM consiste en abordar un proyecto dividiéndolo en distintos ciclos de entrega (conocidos como Sprints). Cada Sprint tiene una serie de trabajos a realizar, y todos los integrantes de cada grupo de cada Sprint se reúnen para colaborar en función de sus conocimientos. Esta metodología está pensada para fomentar la autonomía de los grupos y el compromiso, ya que cada grupo es independiente, con sus distintas metas y plazos marcados por los participantes para llegar al mismo producto final.

Las ventajas de usar Scrum frente a otras metodologías son su cercanía constante con el cliente al mantener reuniones con ellos, el trabajo en equipo, y también es una forma positiva de hacer una auto evaluación, al ver el rendimiento de uno mismo y de su equipo en tiempo real mediante esas reuniones.

Dentro de Scrum, hay una serie de conceptos que se repiten a lo largo de todos sus procesos:

- **Historias de Usuario:** Son una descripción literal de una función que se espera tener del producto final. Es una idea resumida de lo que el cliente espera obtener y, por ende, la comunica este mismo.
- **Sprint:** Son los periodos cortos de trabajo de los proyectos Scrum. El objetivo del sprint es intentar adaptar la forma en la que se trabaja para llegar lo antes posible a un Product Goal (un objetivo marcado del producto). Son periodos cortos donde no se realizan cambios en los proyectos, por lo que no pueden durar mucho tiempo para no arruinar la dinámica del Scrum
- **Tarea:** Son conjuntos de trabajo que un grupo debe realizar. Dentro de los Sprints, hay distintos tipos de tareas a terminar antes de que acabe éste. Se dice que una tarea está acabada o “Done” cuando cumple los requisitos marcados por el cliente y los desarrolladores.
- **Proyecto:** Es la combinación de todos los sprints e historias de usuario realizadas en torno a una misma petición de un cliente.

4 Conclusiones

Odoo ha demostrado ser una herramienta muy potente para la administración de cualquier empresa, ya que, gracias a su capacidad de crear módulos de cero, permite a los usuarios poder darle infinitas utilidades. Teniendo en cuenta que es un software de código abierto y que el usuario no requiere de ningún pago dentro del ERP, se puede considerar uno de los mejores.

A nivel de programación, Odoo dota de una gran gran facilidad para programar módulos de su sistema, utilizando Python como lenguaje de programación y XML como lenguaje de marcas para las vistas. Cualquier cosa necesaria para entender el uso de la arquitectura de datos MVC, el mapeo ORM o cualquier duda posible tiene su respuesta en la documentación de Odoo, haciendo así que sea un sistema muy confiable, sencillo de utilizar y con infinidad de módulos a crear para poder personalizar la experiencia a gusto de la empresa.

5 Bibliografía

Funcionamiento de la metodología SCRUM: <https://ilimit.com/blog/metodologia-scrum/#como-funciona-la-metodologia-scrum>

Definición de historias de usuario: <https://blog.comparasoftware.com/historias-de-usuario-de-scrum-plantilla-y-ejemplos/>

Definiciones de la metodología Scrum: <https://www.scrum.org/> y <https://scrumdictionary.com/>

Definiciones de Odoo: <https://www.sdi.es/tecnologias/odoo/>

Acciones de Odoo:

<https://www.odoo.com/documentation/15.0/es/developer/reference/backend/actions.html>

Enlace al repositorio de GitHub con este módulo:

<https://github.com/ErikAT04/Odoo-SCRUM-Management>