IES RIBERA DE CASTILLA

PROYECTO SGE

CFGS Desarrollo de Aplicaciones Multiplataforma
Informática y Comunicaciones

Desarrollo del módulo “manage” con Odoo ERP

Año: 2024-2025

Fecha de presentación: *dd-mm-2025*

Nombre y Apellidos: Jaime González Bravo
Email: jaime.gonbra@educa.jcyl.es

Índice

Modificar la **portada** con vuestros datos personales.

TÍTULO DEL PROYECTO: "Desarrollo del módulo "manage" con Odoo ERP; para gestionar proyectos usando metodologías ágiles: scrum".

El primer apartado que vamos a incluir es la **INTRODUCCIÓN**: hablar brevemente sobre los sistemas ERP y las metodologías ágiles (en concreto sobre SCRUM).

Segundo apartado: **ORGANIZACIÓN DE LA MEMORIA** (lo vamos a dejar para el final; consistirá en enumerar los apartados de la memoria, con una pequeña descripción de cada uno de ellos).

Tercer apartado: **ESTADO DEL ARTE**. Con los siguientes subapartados:

1. ERP
    a. Definición de los ERP

        ERP (Enterprise Resource Planning) es un tipo de software que las empresas utilizan para gestionar las actividades diarias (contabilidad, gestión de proyectos, etc.). hoy en día, son elementos imprescindibles en la gestión de empresas de cualquier tamaño.

    b. Evolución de los ERPs

        El ERP surge en 1960 cuando se vieron en la necesidad de unificar todos los programas básicos que se hacían antes de la creación de los ERPs.

        Entre la década de los 70, se desarrolla un sistema para gestionar la producción y los procesos de las empresas bajo en nombre de planificación de requisitos materiales (MRP) para posteriormente modificarse a planificación de recursos manufactureros.

        En la década de los 90 el MRP o ERP ya es funcional para controlar inventarios, producciones, gestiones administrativas y RRHH.

        Hacia el 2000, Gartner (el denominado padre de la industria analista moderna) declara que el concepto de ERP ya es un producto. Se incluía el software basado en internet con acceso en tiempo real de la información de los recursos de la

empresa. Desde su nacimiento no ha dejado de evolucionar y mejorar sus recursos y sistema y se ha convertido en un programa clave para la gestión de una empresa.

Desde el 2006 en adelante, con el impulso de la nube, ha convertido el ERP en una herramienta vinculado a la nube. Además, el aumento de aplicaciones sumado, a la mayor calidad de estos se han convertido en una de las mejores herramientas (si no lo era ya).

c. Principales ERP

-SAP: uno de los mejores sistemas ERP del mercado actual. Este ERP tiene un enfoque de innovación continua pues ayuda a crecer sin límites.

- Sage: multinacional británica dedicada a ofrecer soluciones ERP muy avanzadas.

-NetSuite (Oracle): una de las pioneras en la revolución del "cloud computing" siendo la primera empresa de SaaS del mundo.

-Infor: empresa multinacional de corte SaaS. Originalmente se creó para finanzas.

d. ERP seleccionado (Odoo)

Odoo es un software libre de gestión empresarial capaz de cubrir todas las necesidades de tu negocio gracias a la integración de múltiples aplicaciones.

e. Instalación y desarrollo (*formas de instalación, explicando la que se va a usar para desarrollar el proyecto: Docker)*

A la hora de utilizar Odoo, se puede hacer mediante instalarlo en tu máquina y ejecutándolo lo que te creará un servidor en tu máquina.

Otra manera para ejecutar Odoo es mediante su página web que creará un servicio indicándole los datos necesarios de la empresa (la mejor manera de crearlo).

La última manera, es crearlo mediante Docker que es un sistema de contenedores preparado para almacenar (en general) bases de datos de distintos sistemas.

f. Especificaciones técnicas

i. Arquitectura de Odoo

Odoo utiliza el modelo de modelo vista controlador (MVC) que es un modelo de arquitectura de software que propone la creación de 3 componentes distintos.

-Modelo: es la representación de la información que la aplicación utiliza. Se le suele llamar también la lógica de negocio.

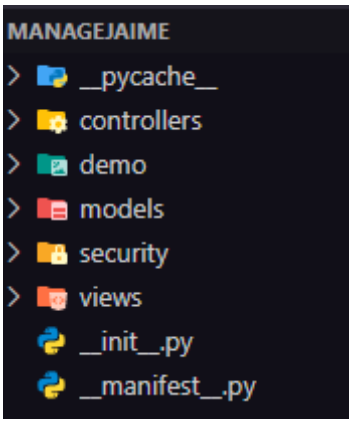
-Vista: presenta la información de la aplicación a través de una interfaz de usuario.

-Controlador: controla la comunicación entre el modelo y responde a los eventos o acciones del usuario.

En el caso de Odoo, el modelo serían los ficheros “Model” de los módulos que es donde se guarda la información, la vista serían los ficheros XML que representan la interfaz de usuario y el controlador serían los ficheros de “Controller” para dar acceso a dicha información.

ii. Composición de un módulo

Un módulo se compone de carpetas tal cual se ven en la foto.



Como podemos apreciar, está dividido en varias carpetas entre las que podemos diferenciar la de controllers, la de models y la de views. Dichas carpetas corresponden al modelo de arquitectura de software visto con anterioridad.

Dentro de controllers, estarán los ficheros que necesitemos para dar acceso a los usuarios.

Dentro de models, estará toda la infraestructura de la lógica de datos de nuestra aplicación.

Dentro de views, estarán todos los ficheros que generan la vista que el usuario verá a la hora de utilizar esta aplicación.

También está la carpeta security en la que daremos permiso a los modelos que creemos para así poder añadir campos a dichas tablas.

2. SCRUM

a. Definición de SCRUM

SCRUM es un marco de trabajo para el desarrollo ágil del software. Es un proceso en el que se aplican de forma regular un conjunto de prácticas para trabajar conjuntamente. Se caracteriza por:

-Adoptar una estrategia de desarrollo incremental.

-Basar la calidad según el equipo asignado.

-Solapar las diferentes fases del desarrollo.

b. Evolución

Se originó al principio de la década de los 80 donde se analizaban el cómo se desarrollaban los nuevos productos de las principales empresas de manufacturación tecnológica. En dicho estudio, se comparó esta forma de trabajo con una formación en específica del rugby que se denominaba Scrum así que de ahí viene el nombre.

Aunque esta forma de trabajo surgió en empresas tecnológicas, se puede emplear en cualquier otro proyecto en los que sus requisitos sean variables y en los que se requiera rapidez y flexibilidad.

En 1995, se presentó “Scrum Development Process”, un conjunto de reglas para el desarrollo de software basados en los principios Scrum.

c. Funcionamiento

La metodología Scrum consiste en abordar cualquier proyecto dividiéndolo en Sprints o partes más pequeñas. Se divide en 5 fases:

- Sprint planning: la planificación del Sprint donde se describen las tareas a las que se asignan a cada miembro del equipo, así como el tiempo que se necesita para finalizarse.

- Scrum team meeting: reuniones diarias para evaluar el trabajo realizado por el equipo de trabajo. En ella, se abordan los problemas presentados o que se pueden llegar a presentar.

- Backlog refinement: el repaso de las tareas realizadas con el fin de evaluar el tiempo y esfuerzo del empleado para cada tarea y para resolver algunos inconvenientes encontrados en el camino.

- Sprint review: reuniones en las que participa el cliente en la que se presenta el trabajo realizado. La presencia del cliente es crucial a la hora de conseguir un feedback real y de calidad.

- Retrospective: la reunión final tras la finalización del proyecto en la que se revisa todo lo acontecido durante el mismo. El objetivo de dicha reunión es adquirir conocimientos y mejorar para futuros proyectos.

d. Principales conceptos *(explicar los principales conceptos: proyecto, historias de usuario, sprint, tarea…)*

Proyecto: consiste en el trabajo a realizar con todo lo que conlleva. Generalmente es de larga duración por lo que se divide en partes llamadas Sprints.

Sprint: se le denomina así al período en el cual se lleva a cabo el trabajo en sí. La duración de este ha de estar definida con la base de la experiencia propia del equipo del desarrollo, aunque esta suele tener de tiempo mínimo una semana y de máximo un mes. Al final de dicho sprint, se deberán de presentar el resultado obtenido. Los sprints se dividen a su vez en tareas.

Tareas: es la división mas pequeña que se puede realizar y comprende el desarrollo al que se le asigna uno o varios desarrolladores. Es la lista de deberes que se tienen que hacer durante la realización de un sprint.

3. Proyecto Manage:
    a. Descripción general del proyecto:
        i. Objetivos:

El proyecto consiste en realizar modelo de Odoo para una gestión de proyectos. El objetivo de dicho proyecto es el de emplear todos los conocimientos aprendidos.

ii. Entorno de trabajo:

El IDE es Visual Studio Code: editor de código fuente desarrollado por Microsoft que permite control de versiones con Git, depuración de código, refactorización de código. Es personalizable debido a su gran cantidad de extensiones y a la posibilidad de modificarle los atajos de teclado, el tema del editor y preferencias. Lo elegí debido a su fácil usabilidad y la posibilidad de añadirle extensiones que faciliten el trabajo con Odoo.

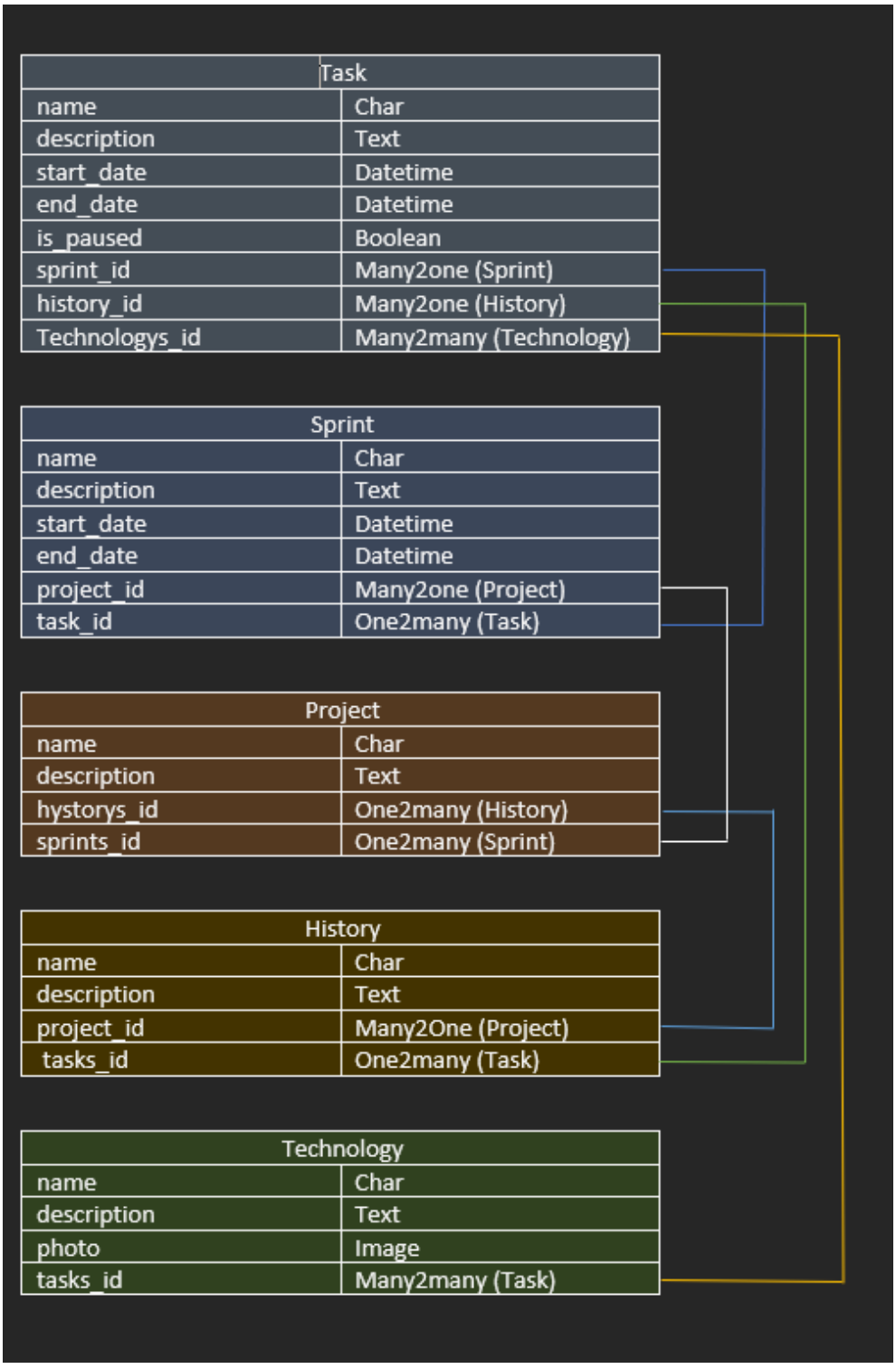
Para desplegar la aplicación, utilizo Docker Desktop: programa de código abierto que automatiza el despliegue de aplicaciones dentro de contenedores de software que proporciona una manera fácil de portar y de ampliar por lo que es muy utilizado hoy en día. Por dentro, Docker utiliza unas características de aislamiento de recursos del kernel de Linux.

Como lenguaje principal, utilizo Python: lenguaje de alto nivel de programación interpretado cuya filosofía hace hincapié en la legibilidad de su código. Soporta la programación orientada a objetos, es dinámico y multiplataforma. Lo he utilizado debido a que Odoo emplea esta tecnología. Además, así afianzo conocimientos en este lenguaje que cada vez de disgusta menos.

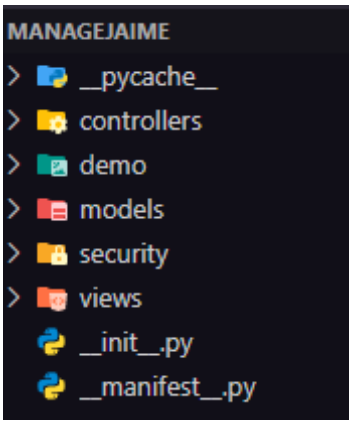
Herramienta principal Odoo: software de ERP integrado que cuenta con la versión comunity de código abierto. Implementa muchas herramientas, desde CRM, facturación, contabilidad, … y todo en una herramienta que se ha convertido en una aplicación de primera necesidad para empresas que tengan un volumen medio-grande.

b. Diseño de la aplicación:

I. Modelo de la base de datos:



II. Organización del proyecto:

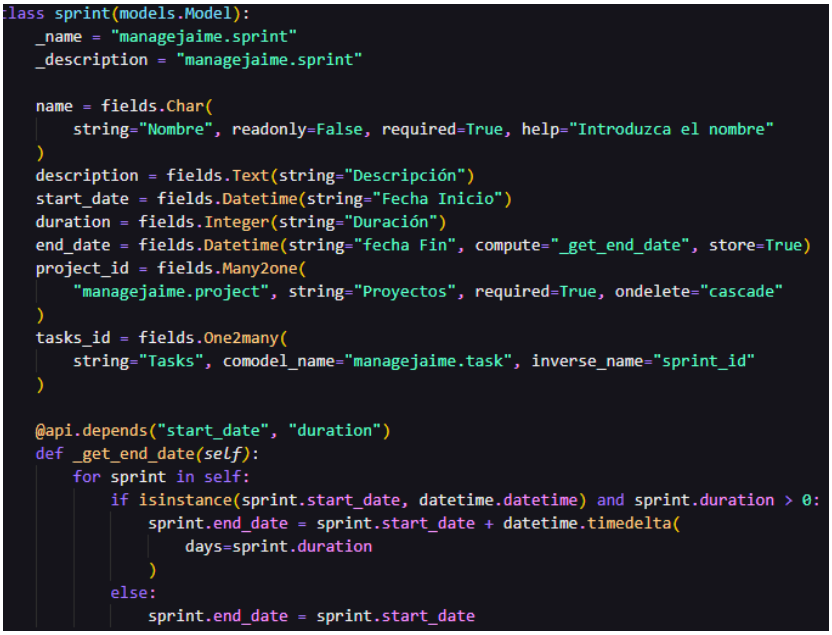


Tal y como se ve en la imagen, utilizo la organización que viene por defecto al crear un módulo en Odoo. Cabe destacar las modificaciones de cada fichero:

En la carpeta models que están todos los modelos (tablas de la base de datos) con todos los campos que he visto necesarios. En él está el fichero model.py en el que esta toda la lógica de negocio. Las clases que utilizo son las de Task, Sprint, Project, History y Tecnology.

```
lass task(models.Model):
    _name = "managejaime.task"
    _description = "managejaime.task"

    code = fields.Char(string="Código", compute="_get_code")
    name = fields.Char(
        string="Nombre", readonly=False, required=True, help="Introduzca el nombre"
    )
    description = fields.Text(string="Descripción")
    start_date = fields.Datetime(string="fecha Inicio")
    end_date = fields.Datetime(string="fecha Fin")
    is_paused = fields.Boolean(string="Pausado")
    sprint_id = fields.Many2one(
        "managejaime.sprint",
        string="Sprint",
        # required=True,
        ondelete="cascade",
        compute="_get_sprint",
        store=True,
    )
    history_id = fields.Many2one(
        "managejaime.history", string="Historias", required=True, ondelete="cascade"
    )
    """project_id = fields.Many2one(
        "managejaime.project", related="history.project_id", readonly=True
    )"""

    technologys_id = fields.Many2many(
        comodel_name="managejaime.technology",
        relation="technology_task",
        colum1="technologys_ids",
        colum2="tasks_ids",
    )

    def _get_code(self):
        for task in self:
            task.code = "TASK_" + str(task.id)

    @api.depends("code")
    def _get_sprint(self):
        for task in self:
            sprints = self.env["managejaime.sprint"].search(
                [("project_id.id", "=", task.history_id.project_id.id)]
            )
            found = False
            for sprint in sprints:
                if (
                    isinstance(sprint.end_date, datetime.datetime)
                    and sprint.end_date > datetime.datetime.now()
                ):
                    task.sprint_id = sprint.id
                    found = True
            if not found:
                task.sprint_id = False
```

La imagen corresponde al modelo Task (tareas). Dicho modelo tiene el campo código que se pondrá automáticamente cuando creemos una nueva tarea. El campo nombre, descripción, fecha de inicio y una fecha de fin y nos dirá si esta pausado o no. Por último, tiene distintas relaciones entre otras tablas como la de id de sprint en los que una tarea se asignara a un sprint al crearse dicha tarea siempre y cuando, la historia está asociada a alguna historia de un proyecto existente. La relación de id de historia en la una historia puede estar vinculado a más de una tarea. La última relación llamada ids de tecnología ya que para una tarea pueden haberse utilizado diferentes tecnologías y/o lenguajes.



```python
class sprint(models.Model):
    _name = "managejaime.sprint"
    _description = "managejaime.sprint"

    name = fields.Char(
        string="Nombre", readonly=False, required=True, help="Introduzca el nombre"
    )
    description = fields.Text(string="Descripción")
    start_date = fields.Datetime(string="Fecha Inicio")
    duration = fields.Integer(string="Duración")
    end_date = fields.Datetime(string="fecha Fin", compute="_get_end_date", store=True)
    project_id = fields.Many2one(
        "managejaime.project", string="Proyectos", required=True, ondelete="cascade"
    )
    tasks_id = fields.One2many(
        string="Tasks", comodel_name="managejaime.task", inverse_name="sprint_id"
    )

    @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
```

La imagen corresponde a la modelo Sprint en el que al igual que tarea, tiene tanto nombre, descripción y fecha de inicio, pero esta agrega tanto el campo duración como el de fecha final. Este último se rellenará automáticamente en función de la fecha de inicio y de la duración del Sprint. Por último, tiene la relación de id de proyecto que vincula el Sprint a un proyecto y la de ids de tareas que relaciona las posibles tareas a un Sprint.

```python
class project(models.Model):
    _name = "managejaime.project"
    _description = "managejaime.project"

    name = fields.Char(
        string="Nombre", readonly=False, required=True, help="Introduzca el nombre"
    )
    description = fields.Text(string="Descripción")
    sprints_id = fields.One2many(
        string="Sprints", comodel_name="managejaime.sprint", inverse_name="project_id"
    )
    historys_id = fields.One2many(
        string="Historias",
        comodel_name="managejaime.history",
        inverse_name="project_id",
    )
```

La imagen corresponde a la clase proyecto en la que tendremos los datos de nombre, descripción, una relación con sprint en la que varios Sprint puedan pertenecer a un solo proyecto y otra relación con historia en la que varias historias pertenecen a un proyecto.

```python
class history(models.Model):
    _name = "managejaime.history"
    _description = "managejaime.history"

    name = fields.Char(
        string="Nombre", readonly=False, required=True, help="Introduzca el nombre"
    )
    description = fields.Text(string="Descripción")
    project_id = fields.Many2one(
        "managejaime.project", string="Proyectos", required=True, ondelete="cascade"
    )
    tasks_id = fields.One2many(
        string="Tareas", comodel_name="managejaime.task", inverse_name="history_id"
    )

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

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

La imagen corresponde a la clase historia en la que tendremos los datos de nombre, descripción, una relación con proyecto en la que a un proyecto se le pueden poner varias historias, una relación con tareas en las que varias tareas pueden pertenecer a 1 historia. Por último, está la relación con tecnologías en la que varias tecnologías pueden ser usadas en varias historias, para ello, empleamos las tecnologías utilizadas en las tareas a las que están asociadas a dicha historia.

```python
class technology(models.Model):
    _name = "managejaime.technology"
    _description = "managejaime.technology"

    name = fields.Char(
        string="Nombre", readonly=False, required=True, help="Introduzca el nombre"
    )
    description = fields.Text(string="Descripción")
    photo = fields.Image(string="Imagen")
    tasks_id = fields.Many2many(
        comodel_name="managejaime.task",
        relation="technology_task",
        colum1="technologys_ids",
        colum2="tasks_ids",
    )
```

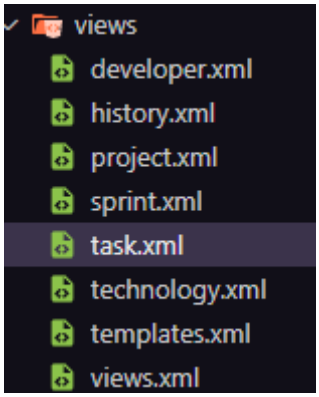
La imagen muestra la clase tecnología en la que tiene los campos de nombre, descripción, foto de la tecnología en cuestión (logo, etc.) y una relación en la que varias tareas pueden utilizar varias tecnologías.

En la carpeta de security, se encuentran los permisos de los usuarios hacia las tablas que acabamos de crear.

ecurity > ir.model.access.csv

```
id,name,model_id:id,group_id:id,perm_read,perm_write,perm_create,perm_unlink
access_managejaime_task,managejaime.task,model_managejaime_task,base.group_user,1,1,1,1
access_managejaime_sprint,managejaime.sprint,model_managejaime_sprint,base.group_user,1,1,1,1
access_managejaime_project,managejaime.project,model_managejaime_project,base.group_user,1,1,1,1
access_managejaime_history,managejaime.history,model_managejaime_history,base.group_user,1,1,1,1
access_managejaime_technology,managejaime.technology,model_managejaime_technology,base.group_user,1,1,1,1
```

En la carpeta de views, se encuentran todas las vistas en formato xml como podemos ver en la imagen inferior. (no pondré las vistas de todos, pero si una para ver que el resto son iguales)



views > project.xml

```xml
<odoo>
    <data>
        <!-- Vista Tree -->
        <record model="ir.ui.view" id="vista_managejaime_project_tree">
            <field name="name">vista_managejaime_project_tree</field>
            <field name="model">managejaime.project</field>
            <field name="arch" type="xml">
                <tree>
                    <field name="name"/>
                    <field name="description"/>
                    <field name="historys_id"/>
                    <field name="sprints_id"/>
                </tree>
            </field>
        </record>

        <!-- Vista Form -->
        <record id="vista__managejaime_project_form" model="ir.ui.view">
            <field name="name">vista__managejaime_project_form</field>
            <field name="model">managejaime.project</field>
            <field name="arch" type="xml">
                <form string="formulario_project">
                    <sheet>
                        <group name="group_top">
                            <field name="name"/>
                            <field name="description"/>
                            <field name="historys_id"/>
                            <field name="sprints_id"/>
                        </group>
                    </sheet>
                </form>
            </field>
        </record>

        <!-- Gestión acciones -->
        <record model="ir.actions.act_window" id="accion_managejaime_project_form">
            <field name="name">Listado de Project</field>
            <field name="type">ir.actions.act_window</field>
            <field name="res_model">managejaime.project</field>
            <field name="view_mode">tree,form</field>
            <field name="help" type="html">
                <p class="oe_view_nocontent_create">
             Project
                </p>
                <p> Click <strong> 'Crear' </strong> para añadir nuevos elementos
                </p>
            </field>
        </record>

        <menuitem name="Manage del alumno JIMYYYYYYYYYYYYY" id="menu_manage_raiz"/>

        <menuitem name="Management" id="Management" parent="menu_manage_raiz"/>

        <!-- acciones -->
        <menuitem name="Project" id="menu_managejaime_projects" parent="Management" action="accion_managejaime_project_form"/>


    </data>
</odoo>
```

En la imagen, podemos ver un ejemplo de vista, en este caso el de proyectos. Dicha vista, está conformada por 2 vistas, la vista “tree” que se utiliza para ver los campos de proyectos ya creados y la vista “form” que se utiliza la hora de añadir o modificar un proyecto. Además, tengo una acción ya que, para poder acceder a estos formularios, creo un menú de opciones. Para ello, al final del fichero, creo los menús ítem necesarios.

Por último, cabe destacar el fichero de manifiesto (_manifest_.py) en el que se deben de añadir todas las vistas o ficheros necesarios para el funcionamiento de la aplicación.

manifest.py

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

    'summary': """
        Short (1 phrase/line) summary of the module's purpose, used as
        subtitle on modules listing or apps.openerp.com""",

    'description': """
        Long description of module's purpose
    """,

    'author': "My Company",
    'website': "https://www.yourcompany.com",

    # Categories can be used to filter modules in modules listing
    # Check https://github.com/odoo/odoo/blob/16.0/odoo/addons/base/data/ir_module_category_data.xml
    # for the full list
    'category': 'Uncategorized',
    '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',
        'views/task.xml',
        'views/sprint.xml',
        'views/project.xml',
        'views/history.xml',
        'views/technology.xml',
        #'views/developer.xml',
    ],
    # only loaded in demonstration mode
    'demo': [
        'demo/demo.xml',
    ],
}
```

III. Ampliación de la aplicación:

Siendo sincero, no tengo nada planeado para ampliar debido a que todavía no tengo el suficiente conocimiento del funcionamiento de la aplicación ya que es el segundo proyecto utilizando Odoo (y Python).

Con tiempo para pensar durante las vacaciones de navidad, he decidido ampliar el proyecto modificando una cosa de Tarea. Para ello, he creado el campo de estado (previamente he borrado el campo boolean de si esta pausado porque no tendría mucho sentido). De esta forma, pongo en funcionamiento los campos de tipo “Selection” que es un campo en el que se permite al usuario de la aplicación elegir un valor de la lista predefinida de opciones. Se define mediante una lista de tuplas (una especie de diccionario de python pero en vez de ser de tipo “clave”: “valor”, lo hace “valor interno”, “texto de muestra”. Por ejemplo, en mi caso el valor interno será de “new” pero a la hora de mostrarlo en la interfaz, pondrá “Nuevo”). Como se puede ver en la imagen, tengo los posibles estados de nuevo, en progreso, pausado, cancelado y completado. Viene por defecto como nuevo y es requerido para crear una nueva tarea.

```python
""" Ampliación de proyecto: creacion del campo estado"""
state = fields.Selection(
    [
        ("new", "Nuevo"),
        ("in_progress", "En Progreso"),
        ("paused", "Pausado"),
        ("completed", "Completado"),
        ("cancelled", "Cancelado"),
    ],
    string="Estado",
    default="new",
    required=True,
)
```

además, he realizado una serie de botones para que el usuario pueda modificar el estado de una tarea al pulsar en cualquiera de ellos. De esta forma, controlo en todo momento el cambio de estado de una tarea.

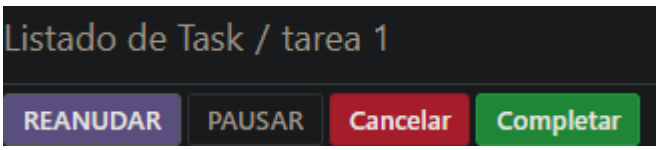
Por último, cree la función de _finalizarTarea() que es un método privado (para evitar poder acceder al fuera de la tarea) que como su nombre indica, pone la fecha de finalización en el momento que se cancela o se completa una tarea.

```python
def reanudarTask(self):
    for task in self:
        if task.state in ["new", "paused"]:
            task.state = "in_progress"

def pausarTask(self):
    for task in self:
        if task.state in ["new", "in_progress"]:
            task.state = "paused"

def cancelarTask(self):
    for task in self:
        if task.state in ["new", "in_progress", "paused"]:
            task.state = "cancelled"
            task._finalizarTarea()

def completarTask(self):
    for task in self:
        if task.state in ["new", "in_progress", "paused"]:
            task.state = "completed"
            task._finalizarTarea()

def _finalizarTarea(self):
    for task in self:
        task.end_date = datetime.datetime.now()
```

también, debido a los cambios que he hecho en el módulo de tareas, he tenido que cambiar su vista (task.xml) en la vista tree también puedes ver el estado actual de la tarea, pero es un cambio mínimo así que decidí no incluirlo aquí. La foto que vemos a continuación es del formulario al añadir/ editar una tarea. Al principio, veremos algo raro. El apartado de header con un atributo que no hemos visto. Se trata de una modificación que realice para mostrar dichos botones solo cuando se edite la tarea. Como podéis ver, cada botón está relacionado a un método y tiene diferentes colores según lo que se quiera realizar. además, decidí que el campo estado, fuera de tipo "statusbar" que vi que era más intuitivo a la hora de visualizarlo y, ya que tenemos botones para modificarlo, no quise que lo pudieras modificar a mano.

```
<form string="formulario_task">
<header attrs="{'invisible': [('id', '=', False)]}">
<!--
Ampliación: botones para poder cambiar el estado de la tarea.
-->
<button name="reanudarTask" type="object" string="Reanudar" state="new" class="btn-primary"/>
<button name="pausarTask" type="object" string="Pausar" state="new,in_progress" class="btn-secondary"/>
<button name="cancelarTask" type="object" string="Cancelar" state="new,in_progress,paused" class="btn-danger"/>
<button name="completarTask" type="object" string="Completar" state="new,in_progress,paused" class="btn-success"/>
</header>
    <sheet>
        <group name="group_top">
            <field name="name"/>
            <field name="description"/>
            <field name="start_date"/>
            <field name="end_date"/>
            <field name="state" widget="statusbar" options="{'clickable': False}"
             attrs="{'invisible': [('id', '=', False)]}"/>
            <field name="sprint_id"/>
            <field name="history_id"/>
            <field name="technologys_id"/>
        </group>
    </sheet>
</form>
```

La imagen es en cuanto a código así que veamos cómo queda en la interfaz de usuario: cuando pulsemos a los botones, se modificará el estado de la tarea salvo que este completada o cancelada que no se podrá modificar ya que se finalizará la tarea.



c. Conclusiones:

Considero que este proyecto puede facilitar el entendimiento o compresión de la herramienta de Odoo y puede ser los primeros pasos para un desarrollador de software con esta herramienta.

d. Bibliografía:

Para infórmame sobre que es Scrum: Qué es scrum y cómo empezar - Atlassian.

Para informarme de Odoo: ERP y CRM de código abierto | Odoo.

Para informarme de la historia y evolución de los ERPs: Odoo.

Para informarme de la historia y evolución de la metodología Scrum: Scrum.

e. Enlace al github:

https://github.com/KonoDIODa13/odoo_dev.git