IES RIBERA DE CASTILLA

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: *15 de enero de 2025*

Nombre y Apellidos: Hugo del Rey Holgueras

Email: hugo.reyhol@educa.jcyl.es

Índice

1 INTRODUCCIÓN:

Los sistemas ERP son aplicaciones usadas en entornos empresariales para la gestión de recursos integrando todos sus datos y procesos.

SCRUM es una metodología ágil para el desarrollo de proyectos que tiene por objetivo entregar valor al cliente de manera rápida y continua, fomentando la colaboración y la adaptación constante.

2 ORGANIZACIÓN DE LA MEMORIA

La memoria se compone de las siguientes partes:

1. **Estado del arte:** Una breve descripción de los ERP, su evolución, instalación, desarrollo y especificaciones técnicas junto la explicación de la metodología de desarrollo ágil SCRUM, su definición, evolución, funcionamiento y principales conceptos.
2. **Descripción del proyecto:** Los objetivos y entornos del trabajo que tiene el proyecto.
3. **Diseño de la aplicación:** La base de datos usada, las partes de las que se compone y la ampliación personal que he realizado al módulo.
4. **Pruebas de funcionamiento:** Los distintos test que he realizado al módulo.
5. **Conclusiones y posibles ampliaciones:** Mi opinión respecto al trabajo y distintas mejoras que podrían implementarse para mejorarlo.

3 ESTADO DEL ARTE

3.1 ERP

3.1.1 Definición de los ERP

Un software ERP es un programa o conjunto de programas que se encargan de las operaciones internas más importantes de una empresa: operaciones relacionadas con la producción, pedidos, gastos, ventas, clientes, nóminas, inventarios...

3.1.2 Evolución de los ERPs

A. Inicios (1960-1980):

- Los primeros sistemas eran soluciones específicas para la planificación de requisitos materiales (MRP).
- Estas herramientas se enfocaban en manufactura y logística, como la gestión de inventarios y órdenes de producción.

B. MRP II (1980-1990):

- Se ampliaron las funciones de los MRP para incluir planificación financiera, dando lugar a los MRP II.
- Se integraron múltiples funciones empresariales (ventas, recursos humanos, finanzas, etc.) en una única plataforma.

C. Era de los ERP modernos (2000-2010):

- Se incorporaron funcionalidades como acceso a tiempo real a la información y recursos basado en internet.
- Aparecieron sistemas especializados para industrias específicas.

D. ERP en la nube (2010-actualidad):

- Se popularizaron soluciones SaaS (Software as a Service) como Odoo.
- Se expande el uso de ERPs gracias a sus menores costes.

3.1.3 Principales ERP

SAP S/4HANA Cloud: Permite a las empresas escalar y adaptarse rápido a las cambiantes condiciones del mercado, su versión privada le permite una gran personalización.

Oracle NetSuite: ERP escalable que proporciona visibilidad, control y agilidad a las organizaciones de rápido crecimiento, ofrece funcionalidades en la nube.

Microsoft Dynamics 365: Conecta sus equipos, procesos y datos en toda la organización para crear experiencias de cliente excepcionales y agilidad operativa.

Odoo: Solución modular de código abierto, adaptable a empresas de todos los tamaños.

3.1.4 ERP seleccionado (Odoo)

Odoo es un ERP que se caracteriza en los siguientes apartados:

- **Modular:** Se compone de distintas aplicaciones las cuales puedes instalar independientemente según las necesite tu empresa.
- **Código abierto:** Permite personalización y desarrollo según las necesidades de cada empresa.
- **Costo:** Es más económico que otros ERP, además cuenta con la versión Community la cual es gratuita.

Por todos estos motivos es el ideal para comenzar a aprender sobre estos y por la cual la hemos seleccionado para desarrollar nuestro módulo.

3.1.5 Instalación y desarrollo

Hay distintas formas de instalar Odoo entre las que destacan tres, instalar el paquete All-In-One para Windows, instalar el código fuente de Odoo sobre un host Ubuntu Server e instalarlo sobre un contenedor Docker. Entre todas estas se va a usar la última y el proceso de instalación es el siguiente:

Paso 1: Instalación de Docker

Descargar la versión de Docker Desktop para nuestro sistema operativo desde la página web oficial.

Doble clic sobre el instalador, una vez instalado reiniciar el ordenador.

Es posible que una vez instalado la aplicación requiera ser abierta por un administrados para poder funcionar correctamente, para configurar esto en Windows podemos abrirlo con clic derecho y pulsando “*abrir como administrador”,* otra solución más cómoda es con clic derecho sobre la aplicación ir a las opciones y en *“Compatibilidad”* seleccionar *“Ejecutar este programa como administrador”*.

Paso 2: Montar el contenedor Docker

Abrir Docker Desktop.

Crear una carpeta en la que van a guardarse los datos de Odoo y crear el archivo docker-compose.yml en ella.

Copiar en el archivo creado lo siguiente:

```
version: "3.8"
services:

  odoo:
    image: odoo:16
    container_name: odoo
    restart: unless-stopped
    links:
      - db:db
    depends_on:
      - db
    ports:
      - "8069:8069"
    volumes:
      - odoo-data:/var/lib/odoo:rw
      - ./config:/etc/odoo:rw
      - ./addons:/mnt/extra-addons:rw
    networks:
      - red_odoo

  db:
    image: postgres:latest
    container_name: container-postgresdb
    restart: unless-stopped
```

```
    environment:
      - DATABASE_HOST=127.0.0.1
      - POSTGRES_DB=postgres
      - POSTGRES_PASSWORD=odoo
      - POSTGRES_USER=odoo
      - PGDATA=/var/lib/postgresql/data/pgdata
    volumes:
      - db-data:/var/lib/postgresql/data
    networks:
      - red_odoo
    ports:
      - "5342:5432"


  pgadmin:
    image: dpage/pgadmin4:latest
    depends_on:
      - db
    ports:
      - "80:80"
    environment:
      PGADMIN_DEFAULT_EMAIL: pgadmin4@pgadmin.org
      PGADMIN_DEFAULT_PASSWORD: admin
    restart: unless-stopped
    networks:
      - red_odoo

volumes:
  odoo-data:
  db-data:

networks:
  red_odoo:
```

Abrir una terminal, ir a la carpeta en la que se encuentra el archivo y ejecutar el comando *docker compose up -d*

Paso 3: Configurar Odoo

Abrir un navegador y buscar localhost:8069

Rellenar los campos y dejar Demo data activado

Iniciar sesión como admin e instalar los módulos ventas, facturación, CRM e inventario

Ir a la carpeta de odoo, entrar en la carpeta config y abrir el archivo odoo.conf

Añadir addons_path =/mnt/extra-addons y guardar los cambios

Reiniciar los contenedores de Odoo

3.1.6 Especificaciones técnicas

3.1.6.1 Arquitectura de Odoo

Odoo se compone de 3 capas:

- **Presentación:** Utiliza HTML, Javascript y CSS
- **Negocio:** Utiliza Python
- **Datos:** Utiliza PostgreSQL

Su arquitectura es MVC (Modelo-Vista-Controlador), los modelos son las clases diseñadas con python, las vistas se escriben en formularios xml y los controladores son los métodos definidos en las clases.

3.1.6.2 Composición de un módulo

Un módulo de odoo se compone de las siguientes partes:

- **Modelos:** Definen la estructura de la base de datos y la lógica de negocio
- **Vistas:** Definen la interfaz de usuario
- **__manifest__.py:** Configuración general del módulo
- **Security:** Maneja los permisos de acceso de los usuarios a la base de datos

3.2 SCRUM

3.2.1 Definición de SCRUM

SCRUM es un marco de trabajo ágil para la gestión de proyectos, especialmente en desarrollo de software. Promueve la colaboración, la flexibilidad y la entrega incremental de valor a través de iteraciones cortas llamadas sprints.

3.2.2 Evolución

Este modelo fue identificado y definido por Ikujiro Nonaka y Takeuchi a principios de los 80. En 1995, Ken Schwaber presentó “Scrum Development Process” un marco de reglas para desarrollo de software, basado en los principios de Scrum.

Con el tiempo, se ha convertido en uno de los marcos más utilizados en industrias tecnológicas y más allá, evolucionando para ser compatible con equipos de diferentes tamaños y objetivos.

3.2.3 Funcionamiento

SCRUM se organiza en ciclos cortos (*sprints*) de 1 a 4 semanas. Cada sprint incluye planificación, desarrollo, revisión y retrospectiva. Los roles principales son el Propietario del producto (*Product Owner*), gestiona el producto teniendo en cuenta el mercado, el Facilitador (*Scrum Master*), elimina obstáculos, asesora y forma al equipo, y el equipo de desarrollo que realiza el trabajo técnico. El progreso se supervisa a través de reuniones diarias (*Scrum*).

3.2.4 Principales conceptos

- **Proyecto:** Trabajo realizado para llevar a cabo un producto, se divide en sprints
- **Historias de usuarios:** Descripciones breves y simples de una funcionalidad deseada del producto
- **Sprint**: Iteración de trabajo de duración fija.
- **Tarea:** Unidad de trabajo más pequeña derivada de una historia de usuario
- **Product Backlog**: Lista priorizada de requisitos del producto.
- **Sprint Backlog**: Tareas seleccionadas del Product Backlog para un sprint.
- **Incremento**: Resultado funcional entregado al final de un sprint.
- **Artefactos**: Elementos clave como el Product Backlog, Sprint Backlog e Incremento.
- **Scrum diario:** Reuniones estructuradas como la planificación del sprint, Daily Scrum, revisión y retrospectiva.

4 Descripción general de proyectos

4.1 Objetivos

Con este proyecto se pretenden alcanzar los siguientes objetivos:

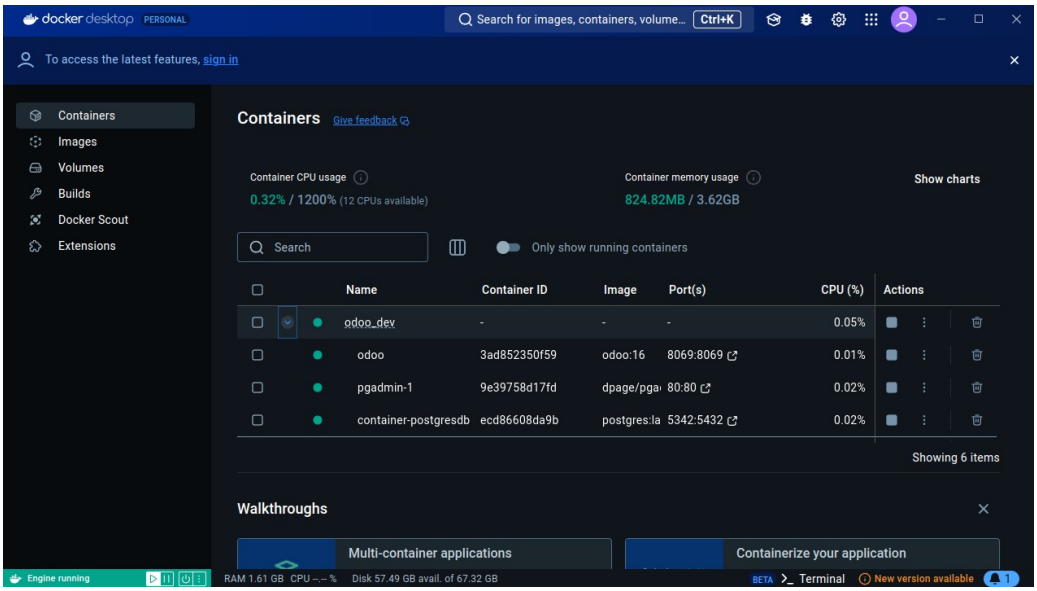
- Crear un módulo para el ERP Odoo mediante el cual gestionar proyectos que sigan la metodología SCRUM
- Aprender a desarrollar distintos módulos para Odoo
- Aprender sobre la metodología de desarrollo ágil SCRUM
- Aprender a crear memorias sobre los proyectos que realicemos

4.2 Entorno de trabajo

Para ejecutar Odoo uso un contenedor Docker el cual estoy ejecutando en Docker Desktop con la configuración mostrada en el apartado 3.1.5.

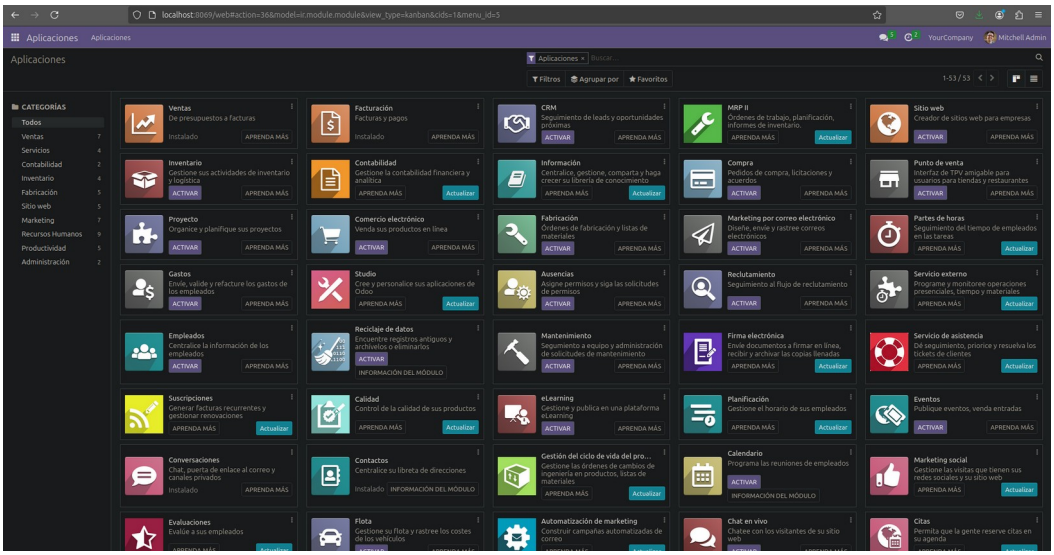
Docker es una plataforma de software que permite desarrollar, empaquetar y ejecutar aplicaciones dentro de contenedores, entornos ligeros, portátiles y consistentes que encapsulan todo lo que una aplicación necesita para funcionar, incluyendo el código, las bibliotecas, las dependencias y el sistema operativo, de manera aislada del resto del sistema, esto nos permite ejecutar una aplicación en una gran variedad de sistemas operativos y equipos sin preocuparnos por las distintas alteraciones que estos puedan causar en nuestra aplicación.

Docker Compose es una herramienta que permite definir y gestionar aplicaciones compuestas por múltiples contenedores, esto nos es de gran utilidad ya que Odoo necesita otros dos módulos para funcionar adecuadamente.



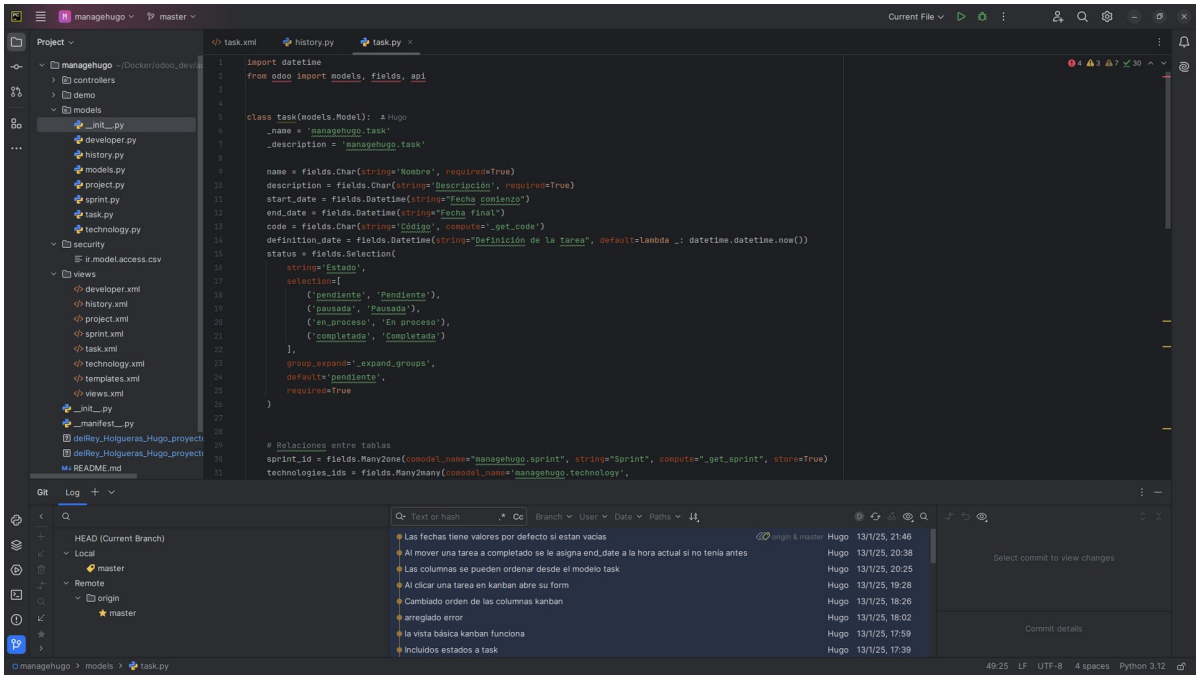
Esta es la interfaz de Docker Desktop, clicando en “odoo” podremos acceder a la terminal del contenedor para ejecutar cualquier comando que necesitemos, como el comando *Scaffold* para crear un módulo nuevo.

Para poder visualizar el entorno gráfico web que usa Odoo necesitaremos usar un navegador web y buscar la dirección del servidor, en nuestro caso *localhost:8069*, yo estoy usando Firefox pero cualquier otro navegador moderno funcionará perfectamente.



Este es el entorno gráfico de Odoo, concretamente la pestaña *Aplicaciones*, desde la cual podremos instalar distintos módulos, tanto oficiales como los creados por nosotros mismo.

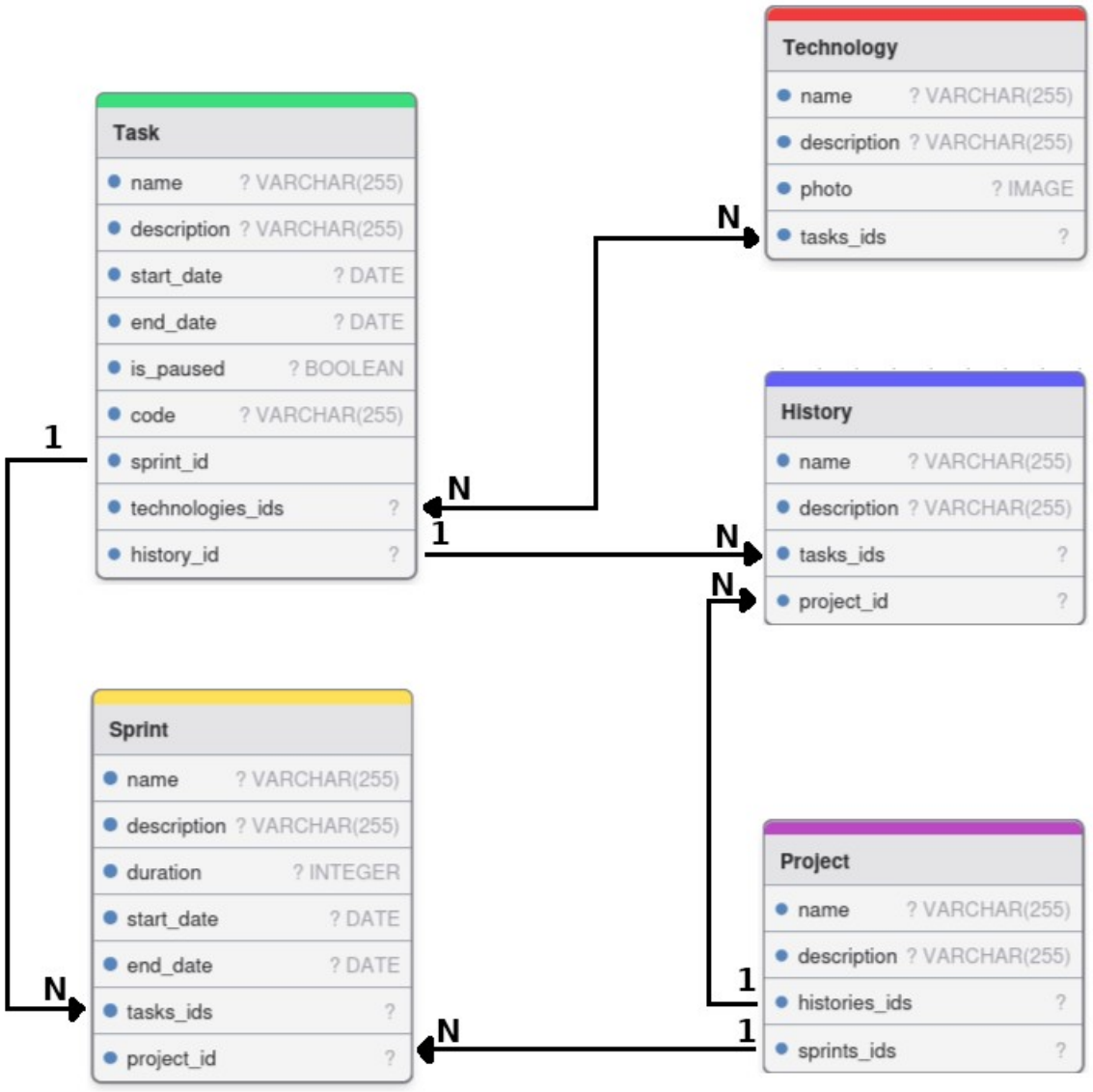
Para escribir el código de las distintas partes del módulo uso PyCharm Community, concretamente la versión 2022.3.1, este es una versión del IDE PyCharm con la diferencia de ser código abierto, este entorno de desarrollo esta pensado para programar Python el cual es el lenguaje usado por el framework de Odoo. Para complementar el IDE he instalado un plugin llamado *Odoo,* el cual añade distintas plantillas, estructuras prediseñadas para módulos, modelos, vistas, y controladores, lo que reduce el tiempo necesario para configurar componentes básicos y garantiza una mayor consistencia en el código.



Este es el entorno gráfico de PyCharm, en él podemos ver la estructura de carpetas del proyecto actual, los detalles del repositorio de git y la zona de edición con el archivo en el que estamos trabajando.

5 Diseño de la aplicación

5.1 Modelo relacional de la BBDD



5.2 Partes del proyecto

El módulo de Odoo se forma de los siguientes componentes:

En la carpeta models he creado un archivo de python para cada clase, las clases son developer, history, project, sprint, task y technology. Cada una de estas clases representan una de las tablas de la base de datos por lo que contienen todas las variables que se almacenarán en esta y la lógica de negocio de la clase es la funcionalidad que tendrán, es decir, el comportamiento de esta representación cuando Odoo esté ejecutándose.

Todos los modelos en Odoo tienen varias cosas en común:

- Incluyen la línea `from odoo import models, fields, api`
- Heredan de models.Model
- Tienen los atributos _name y _description, Strings que comienzan con el nombre del proyecto y seguidos por el nombre del módulo separándolos por un “.”
- Todas las variables que queramos guardar en Odoo deberán pertenecer al paquete fields que fue importado previamente

Estructura básica de un modelo:

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


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

    name = fields.Char(string='Nombre')
    description = fields.Char(string='Descripción')
    start_date = fields.Datetime(string="Fecha comienzo")
```

Como se puede observar esta clase incluye todas los punto dichos previamente, a parte también incluye variables extras las cuales se utilizaran

para generar columnas extras en la base de datos en las que almacenar información que en un futuro nos pueda ser de utilidad.

Campos para relaciones entre modelos:

```
sprint_id = fields.Many2one(comodel_name="managehugo.sprint", string="Sprint", compute="_get_sprint", store=True)
technologies_ids = fields.Many2many(comodel_name='managehugo.technology',
                                    string='Technologies',
                                    relation='tasks_technologies',
                                    column1="technologies_ids",
                                    column2="tasks_ids")

history_id = fields.Many2one(
    comodel_name='managehugo.history',
    string='History',
    required=False)
```

Estas variables son las encargadas de enlazar las distintas tablas de la base de datos entre si, permitiéndonos crear relaciones y lógica más compleja de forma modular.

Existen varios tipos, las Many2one con su respectivo One2many y las Many2many:

Many2one y One2many permiten la relación de varias entidades de una tabla con una entidad de otra tabla distinta, esto nos puede ser útil por ejemplo para gestionar a que historia pertenecen las tareas, ya que varias tareas pueden estar asignadas a la misma historia pero las tareas solo pueden pertenecer a una historia concreta.

Many2many nos permiten la relación de varias entidades de una tabla con varias entidades de otra tabla distinta, esto nos puede ser útil para conocer las tecnologías usadas por las tareas, ya que una tarea puede contener varias de estas y una tecnología puede ser implementada en distintas tareas.

De esta forma podemos realizar varias uniones entre estos módulos que darán más complejidad y funcionalidad a nuestro proyecto.

Los campos de relaciones de la base de datos son gestionados por Odoo, todas necesitarán el nombre del otro modelo con el que se relacionan en *comodel_name*, en el caso de One2many también será necesario incluir

inverse_name el cual contiene en nombre de la variable del otro modelo que hace de unión entre ambos, para las relaciones Many2many también podemos definir *relation*, para nombrar la nueva tabla que se generará en la base de datos, y *column1* y *column2* para nombrar las columnas de los ids que hacen referencia a las tablas originales en la nueva tabla, estos atributos son opcionales y son usados principalmente cuando queremos tener un mayor control sobre la base de datos interna que generará Odoo.

Funciones para la lógica de negocio:

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

@api.depends('code')  Hugo
def _get_sprint(self):
    for task in self:
        sprints = self.env['managehugo.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 lógica de negocio se compone de las distintas funciones que se encuentran en nuestro modelo y lo dotarán de una funcionalidad más compleja, en estas podemos, entre otras cosas, asignarles valores a las variables automáticamente y realizar diversos cálculos, además Odoo tiene incorporadas distintas etiquetas las cuales añaden funcionalidad extra a nuestro código sin la necesidad de programarlo nosotros mismo desde cero, un ejemplo de esto sería la etiqueta *@api.onchange('variable')* la cual nos permitirá ejecutar su función correspondiente cada vez que detecte un cambio en el valor de la variable que hayamos introducido.

Para las vistas he creado distintos archivos, cada uno se relaciona con su modelo especifico, como caso especial al ser task la primera vista que creé también contiene la raíz del menú del módulo.

Los archivos para las vistas se componen de distintas partes:

- Las vistas que van a representar las distintas variables del modelo

```
<record id="vista_managehugo_task_tree" model="ir.ui.view">
  <field name="name">vista_managehugo_task_tree</field>
  <field name="model">managehugo.task</field>
  <field name="arch" type="xml">
    <tree>
      <field name="name"/>
      <field name="description"/>
      <field name="start_date"/>
      <field name="end_date"/>
      <field name="is_paused"/>
      <field name="sprint_id"/>
      <field name="technologies_ids"/>
    </tree>
  </field>
</record>
```

En Odoo he trabajado con tres tipos de vistas:

La vista tree nos permite mostrar la información de las entidades en filas de forma similar a una tabla de datos, es la vista más básica que tiene Odoo

La vista form nos permite modificar los valores de las entidades y guardarlos en la base de datos, también es la que se abre cuando queremos crear una entidad nueva para poder introducir sus datos iniciales

La vista kanban es mucho más personalizable, en ella podemos mostrar la información de las entidades en estructuras a las cuales daremos forma con distintos componentes similar a como lo haríamos en HTML. También podemos programar distintos comportamientos como simplemente mostrar las entidades como harías con una cartelera de películas o organizarlas por alguna de sus variables por columnas y desplazarlas entre estas mediante *drag and drop* y Odoo se encargará de modificar esta misma variable automáticamente.

- Las acciones que pueden ejecutar las distintas vistas

```
<record id="accion_managehugo_task_form" model="ir.actions.act_window">
  <field name="name">Tasks</field>
  <field name="type">ir.actions.act_window</field>
  <field name="res_model">managehugo.task</field>
  <field name="view_mode">tree,form</field>
  <field name="help" type="html">
    <p class="oe_view_nocontent_create">
      Tasks
    </p>
    <p>
      Click <strong> 'Crear' </strong> para añadir nuevos elementos
    </p>
  </field>
</record>
```

Este fragmento de código se encarga de activar las distintas vista de las que hemos hablado anteriormente, esto hace que el usuario pueda cambiar entre ellas si lo desea.

También podemos incluirle apartados extra, en el ejemplo superior en caso de no detectar ninguna entidad para la vista actual mostrará un mensaje explicándole al usuario cómo crear una nueva.

- Los distintos apartados de los que se compone el menú

```
<!-- MENÚ RAÍZ -->

<menuitem name="Manage de Hugo" id="menu_managehugo_raiz"/>

<!-- SEGUNDO NIVEL -->

<menuitem name="Management" id="menu_managehugo_manage" parent="menu_managehugo_raiz"/>

<!-- ACCIÓN -->

<menuitem name="Tasks" id="menu_managehugo_task" parent="menu_managehugo_manage"
          action="accion_managehugo_task_form"/>
```

Tendremos un menú superior para cambiar entre las distintas vistas que hayamos creado, en este caso existirá un apartado principal del que colgaran los distintos elementos del menú, esta será la raíz del menú, a esta raíz le podremos añadir distintos apartados que queramos que el usuario pueda visitar, para ello le asignaremos como *parent* el menú raíz y una *action* con el id de la vista correspondiente, convirtiéndolo en un elemento secundario, también podemos incluir submenús si a uno de los elementos de segundo nivel le añadimos más elementos, esto lo convertirá en un desplegable y mostrará todas las opciones que contenga cuando el usuario lo seleccione.

El archivo *ir.model.access.csv* contiene los permisos que tienen los distintos usuarios para interactuar con la base de datos, al ser un módulo desarrollado con fines educativos todos los usuarios tienen acceso total. Sus valores se separan por comas y por filas.

Como se puede ver en la parte superior de la imagen el orden en el que se especificara la información es el siguiente:

- **id:** El identificador de la regla de acceso que queremos modificar
- **name:** Es el nombre legible de la regla de acceso.
- **model_id:id:** Especifica el modelo al que aplica esta regla de acceso.
- **group_id:id:** Indica el grupo de usuarios al que se aplica esta regla de acceso.
- **perm_read:** Define si los usuarios del grupo pueden leer registros del modelo.
- **perm_write:** Define si los usuarios del grupo pueden modificar registros existentes.
- **perm_create:** Define si los usuarios del grupo pueden crear nuevos registros en el modelo.
- **perm_unlink:** Define si los usuarios del grupo pueden eliminar registros del modelo.

```
id,name,model_id:id,group_id:id,perm_read,perm_write,perm_create,perm_unlink
access_managehugo_task,managehugo.task,model_managehugo_task,base.group_user,1,1,1,1
access_managehugo_sprint,managehugo.sprint,model_managehugo_sprint,base.group_user,1,1,1,1
access_managehugo_project,managehugo.project,model_managehugo_project,base.group_user,1,1,1,1
access_managehugo_history,managehugo.history,model_managehugo_history,base.group_user,1,1,1,1
access_managehugo_technology,managehugo.technology,model_managehugo_technology,base.group_user,1,1,1,1
```

El archivo *__init__.py* situado en la carpeta *models* le indica al framework de Odoo a que modelos puede acceder cuando se inicialice el módulo.

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

El archivo *__manifest__.py* contiene los metadatos que describen y configuran el módulo, pero la que es esencial modificar es la lista data, en la cual he añadido todas las vistas que he creado ya que esta lista indica que archivos deben cargarse al iniciar el módulo.

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

5.3 Ampliación del proyecto

He decidido llevar a cabo una ampliación sobre el modelo y vista para task ya que es uno de los apartados en los que más tiempo va a estar un usuario.

Mi objetivo con esta mejora es convertir la vista para tareas en una vista kanban en la que el usuario podrá mover las tareas entre las distintas columnas modificando a la vez su estado y otras variables.

Para mejorar el modelo task he eliminado la anterior variable booleana “*is_paused*” para sustituirla por un campo de selección, lo cual permite a las distintas tareas tener varios estados entre los que pueden cambiar.

Me he decantado por solo incluir cuatro para no recargar la interfaz, las cuales son “*Pendientes*”, *“Pausada”*, *“En proceso”* y *“Completada”*.

```
status = fields.Selection(
    string='Estado',
    selection=[
        ('pendiente', 'Pendiente'),
        ('pausada', 'Pausada'),
        ('en_proceso', 'En proceso'),
        ('completada', 'Completada')
    ],
    group_expand='_expand_groups',
    default='pendiente',
    required=True
)
```

Las variables de *Selection* son:

- **selection:** La lista de distintos estados que puede tener una tarea, es una tupla en la que el primer valor representa el estado y el segundo es el nombre que veremos en la vista
- **default:** Establece el valor por defecto que tomará la tarea al crearla
- **required:** Establece el campo como obligatorio

- **group_expand:** Es un método que devuelve el orden en el que se van a ordenar las columnas cuando se cargue la vista kanban

El método *_expand_groups* devuelve una lista con los valores de los estados en el orden en el que queramos representarlos y también se asegura de que todas las columnas se muestre incluso cuando no tengan registros asignados.

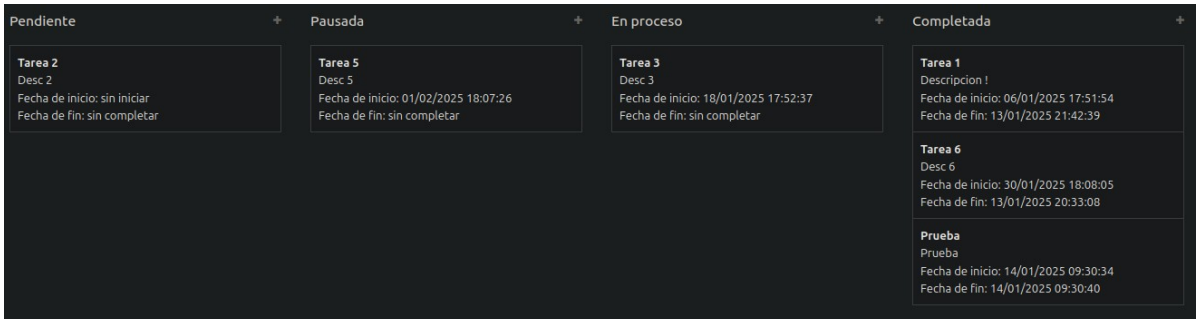
```
@api.model  Hugo
def _expand_groups(self, states, domain, order):
    return ['pendiente', 'pausada', 'en_proceso', 'completada']
```

El modelo también tiene un método que detecta cuando se realizan cambios en el campo *estado* y reacciona a estos actualizando las fechas de inicio y fin de la tarea automáticamente a la fecha en el momento del cambio.

Para evitar eliminar un valor importante se comprueba que la fecha no tuviese un valor antes de asignarle el nuevo, para asignar la fecha de fin espera a detectar un cambio hacia *“completado”* y para la fecha de inicio espera a que deje de estar en *“pendiente”*, de esta forma el usuario final puede desplazar la tarea en la vista kanban cómodamente sin tener que entrar en el menú de edición para cambiar las fechas.

```
@api.onchange('status')  Hugo
def _onchange_status(self):
    for task in self:
        if task.status == 'completada' and not task.end_date :
            task.end_date = datetime.datetime.now()
        if task.status != 'pendiente' and not task.start_date:
            task.start_date = datetime.datetime.now()
```

Para poder mostrar las tareas he hecho una vista kanban en la que se dividen en distintas columnas dependiendo de sus estados y la he establecido como vista principal.



Las características principales son la capacidad de mover las tareas entre estados usando un sistema *drag and drop*, poder crear tareas nuevas en cada columna usando el simbolo “+” en cada una de ellas y poder abrir la vista *form* con la información de una tarea al clicar sobre ella.

El código de la vista es el siguiente:

```
<record id="vista_managehugo_task_kanban" model="ir.ui.view">
  <field name="name">vista_managehugo_task_kanban</field>
  <field name="model">managehugo.task</field>
  <field name="arch" type="xml">
    <kanban class="o_kanban_view" default_group_by="status">
      <field name="status"/>
      <templates>
        <t t-name="kanban-box">
          <div t-attf-class="oe_kanban_global_click">
            <div class="o_kanban_details">
              <strong><field name="name"/></strong>
              <div><field name="description"/></div>
              <div>
                <span>
                  Fecha de inicio: <field name="start_date"/><span t-if="!record.start_date.value">sin iniciar</span>
                </span>
              </div>
              <div>
                <span>
                  Fecha de fin: <field name="end_date"/><span t-if="!record.end_date.value">sin completar</span>
                </span>
              </div>
            </div>
          </div>
        </t>
      </templates>
    </kanban>
  </field>
</record>
```

Los puntos más importantes para que la vista funcione correctamente son:

- **default_group_by:** Especifica la variable del modelo que servirá de referencia para ordenar las columnas

- **t-attf-class:** Le da a los contenedores la posibilidad de abrir la tarea en su vista form
- **o_kanban_details:** Detalla la forma en la que se va a mostrar en el contenedor la información
- **t-if:** Muestran ciertas partes si su condición se cumple, son *ifs* de los que dispone el framework de Odoo para las vistas

Para poder mostrar la vista kanban como la principal se tiene que incluir en el *view_mode* la primera.

```
<record id="accion_managehugo_task_form" model="ir.actions.act_window">
  <field name="name">Tasks</field>
  <field name="type">ir.actions.act_window</field>
  <field name="res_model">managehugo.task</field>
  <field name="view_mode">kanban,tree,form</field>
```

6 Pruebas de funcionamiento

Hay varias pruebas que hay que realizar para comprobar el correcto funcionamiento del módulo, la primera es el orden en el que hay que empezar un proyecto para que todos los campos y relaciones automáticas se creen adecuadamente, el orden a seguir es:

1. Crear un **proyecto** con su nombre
2. Crear una **historia** con su nombre y descripción y seleccionar el **proyecto** anterior
3. Crear un **sprint** con su nombre, duración y fecha de comienzo y seleccionar el **proyecto**

Al hacerlo de esta forma a todas las tareas nuevas se le asignan automáticamente al sprint activo.

La segunda prueba a realizar es el introducir valores distintos a los tipos de las variables, en este caso Odoo se encarga automáticamente de detectar que el tipo de valor es erróneo y registra el valor en la base de datos.

La tercera prueba es comprobar que todas las columnas de la vista kanban se mostrasen incluso si no tenían elementos, al principio no funcionaban pero al asignarle el *group_expand* al modelo las columnas se visualizan incluso estando vacías.

La cuarta es comprobar que cuando una tarea tenga una o varias fechas sin valor se muestre un mensaje en la vista indicándolo y que al moverla entre columnas solo se actualice al nuevo valor si este no tenía valor previamente, esto se controla desde las funciones del módulo.

Estas son algunas de las pruebas que he realizado después de programar para comprobar que el código funcionase correctamente antes de subirlo a GitHub.

7 Conclusiones y posibles ampliaciones

Ha sido un proyecto en el que me ha gustado trabajar y he aprendido muchas cosas tanto de Odoo como de SCRUM, ambas son importantes en entornos de trabajos profesionales por lo que además todo lo que he aprendido también me será de utilidad en mi vida laboral.

Odoo me ha parecido una herramienta muy útil y versátil, además el desarrollo de módulos para este es más sencillo de lo que parece al principio, por lo que si es necesario incluir alguna función al flujo de trabajo y ningún módulo oficial dispone de ella, desarrollar un nuevo módulo o modificar uno preexistente para agregarla son opciones viables.

Respecto a la metodología SCRUM, aunque me parezca algo caótica, entiendo perfectamente su uso y por qué se aplica en muchos desarrollos actuales.

Como pega diría que desarrollando este módulo muchas veces no tenía del todo claro por qué hacíamos ciertas relaciones o restricciones respecto a los modelos ya que era la primera vez que escuchaba con tanto detalle el funcionamiento del SCRUM, esto hacía que me costase seguir el ritmo de ambas cosas a la vez y pienso que posiblemente me hubiese resultado más sencillo aprender ambas partes por separado.

Me habría gustado realizar varías ampliaciones, entre ellas:

- Poder asignarle tareas a los desarrolladores
- Incluir una pestaña en la que cada desarrollador pudiese ver todas sus tareas pendientes
- Incluirle prioridades a las tareas y poder filtrar por estas
- Distintos estados adicionales, como “pendiente para revisión”, distintos a las etapas de las tareas
- Una barra de progreso de las tareas totales
- Poder notificar a los desarrolladores cuando se les asigne una nueva tarea o se le cambie la prioridad a una que tuviesen pendiente

Estas eran algunas de las ideas que se me ocurrieron para hacer la ampliación del proyecto.

8 Bibliografía

- Evolución ERP:

https://www.terabyte2003.com/erp-origen-evolucion/

https://www.calipso.com/articulos/historia-del-erp-origen-linea-de-tiempo-y-futuro/

- Odoo:

Temario dado en clase

- SCRUM:

https://www.atlassian.com/es/agile/scrum

https://es.wikipedia.org/wiki/Scrum_(desarrollo_de_software)