RESUMEN 1ER TRIMESTRE

Sistemas de Gestión Empresarial

19 DE NOVIEMBRE DE 2024

ERIK AMO TOQUERO

UD3 – Python

(Más ampliación en los Mini Apuntes de Python)

Características de Python 3

Python es un lenguaje de programación nacido en los 90 por Guido Van Rossum. Recibió su nombre por Monty Python's Flying Circus (Del grupo Monty Python). Actualmente, se utiliza Python 3, el cual ofrece:

- **Sintaxis Sencilla:** Los programas de Python se parecen mucho al pseudocódigo inglés
- **Potente:** Se puede implementar muchas acciones en pocas líneas
- **Interpretación:** Se ejecuta instrucción a instrucción, por lo que es más ágil
- **Anarquía:** Es un lenguaje en el que no hay que declarar nada
- **Curva de Aprendizaje Suave:** Por su sencillez, se utiliza para aprender a programar

Se utiliza para servidores, IA, big data y otros campos por su gran optimización

Cómo trabajar en Python

El código de Python se puede trabajar de 2 formas:

- **Online:** Desde un entorno de desarrollo web como Jupyter
- **Local:** Desde un IDE como VSC o PyCharm

Instalación de Python en Windows para Odoo:

1. Comprobar la versión de Python que usa Odoo (Comunidad – Documentación, Guías de Usuario)
2. Ir a la web oficial de Python y descargar la versión 3.X (Nada de 2.x)
3. Descargar e Instalar asegurando elegir la opción *Add python.exe to PATH*
4. Verificar su correcta instalación con el comando de cmd *python -V*

Instalación y Configuración del IDE de Visual Studio Code:

1. Descargar el IDE en code.visualstudio.com
2. Descargar la versión System Installer
3. Aceptar las casillas "Registrar Code como editor para tipos de archivo permitidos" y "Agregar a Path"
4. Descargar la extensión de Python para Visual Studio code

Instalación y Configuración del IDE de PyCharm

1. Acceder a la web de JetBrains de PyCharm y descargar la edición de Community (si no se cuenta con suscripción)
2. Instalar la aplicación marcando "Create Associations" y "Add 'bin' to PATH"

Estructura de un programa Python

La estructura de un programa va a ser normalmente la siguiente:

- **Imports:** Importe de librerías externas
- **Variables:** La creación de las variables que se vayan a usar a lo largo del proyecto
- **Funciones:** La creación de todas las funciones que se vayan necesitando

Configuración de un Proyecto en VSC

1. Crear o abrir un directorio para el proyecto
2. Escribir virtualenv venv (quizá hace falta instalar virtualenv con pip)
3. Abrir el entorno virtual con .\venv\Scripts\activate.ps1
4. Crear los ficheros y carpetas que se vayan a usar

Configuración de un Proyecto en PyCharm

1. Crear o abrir un proyecto e indicar donde se va a guardar
2. Usar Virtualenv para los entornos
3. Poner de intérprete base el descargado
4. Crear los ficheros y carpetas que se vayan a usar

Aspectos de Python

- No se marca el final de la línea (!;)
- Todo se marca por indentación (!{})
- No se declaran las variables obligatoriamente
- Distinguir mayúsculas y minúsculas, además de no usar números, rayas y espacios
- Usar parámetros opcionales dando un valor por defecto
- Comentar con # una línea y “””comentario””” para varias
- Se recomienda snake_case (todo en minúsculas y unido por guiones
- Se puede correr el programa con python archivo desde CMD y con la terminal incorporada de cada IDE

UD4 – IDE y Primer Módulo de Odoo

Introducción

Con Odoo instalado, se pueden desarrollar módulos que amplían su funcionalidad. Se recomienda realizar una configuración previa en el sistema ERP y en las herramientas de desarrollo asociadas. Una vez hecho, se empezará a editar un módulo que de primeras no hace nada.

Arranque de Odoo y creación de directorios

Para arrancar Odoo usando Docker es necesario tener un directorio para incluir los módulos de Odoo. Hará falta Docker Compose: Una herramienta que permite definir contenedores que interactúan entre sí. Se basa en un archivo llamado Docker-compose.yaml, que define:

- **Servicios:** Cada contenedor con sus configuraciones
- **Redes** de comunicación entre contenedores
- **Volúmenes:** Almacenamiento compartido entre los sistemas virtuales y el físico.

```
version: "3.8" #Versión de Compose

services: # Servicios


  odoo: # Odoo

    image: odoo:16 #Imagen que se va a usar, con su versión
    container_name: odoo # Nombre del contenedor
    restart: unless-stopped # Si está esta opción, el contenedor no se
reiniciará hasta que se reinicie
    links: # Enlaces
      - db:db # Enlaza la base de datos de PosgreSQL
    depends_on: # Dependencias
      - db # El contenedor no funcionará sin PosgreSQL
    ports: # Puertos que ocupa (máquina:contenedor)
      - "8069:8069" # Puerto por defecto de Odoo
    volumes: # Volúmenes físicos
      - odoo-data:/var/lib/odoo # Datos persistentes
      - ./config:/etc/odoo # Datos personalizados
      - ./addons:/mnt/extra-addons # Addons (módulos)
    networks: #Redes personalizadas
      - red_odoo



  db: # Base de Datos de PosgreSQL
    image: postgres:latest # Imagen del contenedor de PosgreSQL y versión
```

```
    container_name: container-postgresdb # Nombre del contenedor
    restart: unless-stopped # No se reinicia hasta que se pare
    environment: # Variables de entorno
      - DATABASE_HOST=127.0.0.1 # Host de la bd
      - POSTGRES_DB=postgres # Nombre de la base
      - POSTGRES_PASSWORD=odoo # Contraseña de usuario
      - POSTGRES_USER=odoo # Nombre de usuario
      - PGDATA=/var/lib/postgresql/data/pgdata # Ubicación de los datos
     volumes: # Volúmenes que ocupa
      - db-data:/var/lib/postgresql/data # Datos persistentes
    networks: # Red personalizada
      - red_odoo
    ports: # Puerto que ocupa (local:contenedor)
      - "5342:5432"


  pgadmin: # Gestor de BD de PosgreSQL
    image: dpage/pgadmin4:latest # Imagen del contenedor y versión
    depends_on: # Dependencia
      - db # Depende de la base de datos para iniciarse
    ports: # Puertos que ocupa (local:contenedor)
      - "80:80"
    Environment: # Variables de Entorno
      PGADMIN_DEFAULT_EMAIL: pgadmin4@pgadmin.org # Correo de acceso
      PGADMIN_DEFAULT_PASSWORD: admin # Contraseña
    restart: unless-stopped # Solo se reinicia si se apaga
    networks: # Redes personalizadas
      - red_odoo

volumes: # Volúmenes creados
  odoo-data: # Odoo
  db-data: # Posgre y PGAdmin

networks: # Redes personalizadas
  red_odoo:
```

Los comandos principales de Docker Compose son **up** (levanta la aplicación), **down** (elimina), **logs** (muestra los logs) y **exec** (ejecuta un comando)

Para crear y levantar los contenedores de Odoo hace falta:

1. Instalar Docker y Docker Compose (en Windows ya viene instalado con Docker)
2. Abrir Docker Desktop
3. Configurar en una carpeta el archivo docker-compose.xml
4. Levantar el contenedor en la carpeta anterior con docker-compose up -d

Errores de Docker

1. **WSL Installation is Incomplete:** Docker utiliza Windows Subsystem for Linux, una opción de Windows que permite tener un subsistema de Linux trabajando y ejecutando acciones. Para instalar Docker, es necesario actualizar, habilitar y reiniciar WSL 2.
2. ***deploying WSL2 distributions ensuring main distro is deployed: checking if main distro is up to date: checking main distro bootstrap version: getting main distro bootstrap version: open \\wsl$\docker-desktop\etc\wsl_bootstrap_version: The network name cannot be found. checking if isocache exists: CreateFile \\wsl$\docker-desktop-data\isocache\: The network name cannot be found ->*** Ejecutar wsl –list –verbose, wsl –set-default-version 2 y wsl—shutdown

Ante la duda **arrancar siempre Docker como administrador**

Si tu usuario principal no es administrador por algún motivo, hacer lo siguiente:

- Entrar como usuario administrador en usuarios y grupos locales – grupos – Administradores y añadir al usuario.
- Ponerle en el grupo docker-user
- Reiniciar y entrar con tu usuario
- Usar Docker (docker compose up -d)

Entornos de Desarrollo para Odoo

Los dos más potentes son Visual Studio Code y PyCharm, además de ser necesario el Control de Versiones.

- Visual Studio Code: Muy potente, con muchos plugins para ampliar su funcionalidad, además de un uso extendido del control en versiones
- PyCharm: No reconoce los elementos de Odoo, pero sí Python. PyCharm permite el uso de Git desde su entorno

Extra: Automatizar reinico de Odoo

Hará falta automatizar Odoo para reiniciarse automáticamente si queremos que sea sencillo que se cierre y se abre, sin causar problema. Para ello, se usará PowerShell:

1. Cambiar ExecutionPolicy a RemoteSigned
2. Abrir PowerShell ISE
3. Meter el siguiente código y guardar para su próximo uso:

```
docker stop container-postgresdb
docker stop odoo
docker start container-postgresdb
docker start odoo
Start-Process chrome.exe http://localhost:8069/web?debug=1
```

UT5: Desarrollo de Módulos de Odoo: Modelo y Vista

(Más info en Mini Apuntes Odoo 1)

Odoo es un ERP-CRM de código abierto que se distribuye bajo despliegue on-premise y SAAS (Proporciona servicio a la nuve)

Odoo usa una arquitectura de 3 capas: Presentación (CSS, HTML, JS), Negocio (Python) y datos (PostgreSQL). Odoo se consta de un módulo base y todos los módulos que cuelgan de este.

Nada más instalar Odoo, se tiene acceso al backend y al desarrollo por medio de módulos como Compras, CRM y Contabilidad. No es necesario modificar el código de Odoo, solo crear un módulo.
Odoo facilita el desarrollo de módulos gracias a su framework de programación, que permite crear módulos con varias técnicas:

- ORM -> Mapeo de Objetos Relacionales: Convierte las tablas de bases de datos en modelos de objetos en Python
- Arquitectura Modelo-Vista-Controlador
    - Modelo: los objetos de Python que interactúan con la base de datos
    - Vistas: XML con listas, formularios y menús que se formarán por medio del framework de Odoo
    - Controlador: Métodos que proporcionan la lógica
- Tenencia Múltiple: Un único servidor da servicio a muchos clientes
- Diseño de Informes
- Traducción automática

Base de Datos

Gracias al ORM, no hay un diseño de base de datos, sino que los modelos que se creen con models.Model serán mapeados automáticamente a una tabla). Como puede haber tablas muy distintas, Odoo proporciona el “Modo Desarrollador”, que permite saber el modelo y campo de los módulos que creamos.

Composición de un módulo:

Los módulos son directorios compuestos de distintos ficheros:

- **Ficheros Python** para definir módulos y controladores
- **Ficheros XML** Para las vistas y datos estáticos
- **Ficheros estáticos** que deben cargarse por la interfaz web
- **Controladores web** para gestionar las peticiones
- **Fichero Manifest:** Contiene toda la información necesaria para interpretar los ficheros de un directorio.
- **Fichero Init:** Directorios de Python a cargar al principio

Ficheros generados al principio:

- **models/models.py:** Ejemplos de la estructura de un modelo
- **views/views.xml:** Ejemplos de la estructura de una vista
- **demo/demo.xml:** Datos de demostración
- **controllers/controllers.py:** Ejemplos de controladores
- **views/templates.xml:** Ejemplos de vistas qweb
- **__manifest__.py:** Manifiesto del módulo, con los datos a cargar.

Creación de un módulo nuevo: odoo scaffold nombremodulo /rutamodulo

Cada vez que se haga algún cambio: Reiniciar Odoo, Actualizar Lista de Aplicaciones y buscar el nombre del módulo.

Modelos

Los modelos definen estructuras para manejar y almacenar datos del módulo. Heredan la clase models.Model para adquirir ciertas propiedades del ORM. Hay 4 tipos de modelo.

- **Model:** Modelo más común y usado
- **TransientModel:** Datos temporales y que se borran periódicamente
- **AbstractModel:** Modelo abstracto que será heredado por múchos otros modelos.

Los módulos tendrán que aparecer registrados en el __init_.py de models.

Cuando se active el módulo, se podrá mirar la estructura de la tabla en ModoDesarrollador->Ajustes->Menú Técnico->Modelos

Vistas

En el esquema Modelo-Vista-Controlador, la vista se encarga de interactuar con el usuario. Es la interfaz gráfica que el usuario utiliza para gestionar la información. Tiene varios elementos para funcionar:

- Definiciones de vistas: Las propias definiciones de las vistas en “ir.ui.view”. Tienen los fields que se van a mostrar, su comportamiento y su aspecto
- Menús: Están distribuidos de forma jerárquica y se guardan en el modelo ir.ui.menu
- Actions: Las acciones enlazan una acción del usuario con una llamada al servidor “ir.ui.action”

La vista define cómo se presentan y visualizan los datos del módulo, además de definir los modelos que aparecen y los iconos del módulo.

Dentro de un módulo de Odoo puede haber varias vistas, de varios tipos:

- Vista Menú: Se puede definir un menú en la barra superior para acceder al resto de vistas (“ir.ui.menu”)
- Vista Formulario: Mediante esta vista permite al usuario crear, modificar y/o borrar datos (“ir.ui.view”)
- Vista tree: Es un listado de todos los registros en el modelo. Se accede a su formulario clicando en uno (“ir.ui.view.”)
- Vista Kanban: Presenta los registros en tarjetas rectangulares, teniendo así una representación virtual. Si se hace clic en ellos, se accede a su interior (“ir.ui.view”)

Si hay una vista tree y Kanban en el mismo árbol, se podrá alternar de vista en el menú superior

Todas las vistas estarán entre <odoo><data></data></odoo>

Cuando se haga un cambio en una vista, habrá que reiniciar odoo y actualizar la aplicación

Nivel de los menús

Como se mencionó previamente, los menús son elementos jerárquicos en una vista. Esta jerarquía se marca con los parents, y tiene un total de 3 niveles:

1. **Raíz:** Rama principal, lo que se ve al dar al menú de aplicaciones
2. **Segundo Nivel:** Rama que cuelga de Raíz, marca pestañas del menú principal
3. **Tercer nivel:** Rama que cuelga de la segunda, marca subapartados con acciones.

<menuitem id=”idMenu” name =”nombreMenu” sequence=”” [parent=”id padre”] [action = “actionid”] [web_icon=”urlicono”]/>

Permisos

Dentro de Odoo, se pueden ver los distintos usuarios y grupos en Ajustes -> Usuarios -> Grupos. Cada uno de ellos tiene 4 tipos de permiso aplicable:

- **perm_read:** Lectura: Puede ver los registros
- **perm_write:** Escritura: Puede modificar
- **perm_create:** Creación: Puede añadir registros
- **perm_unlink:** Borrado: Puede quitar registros

Para editar estos permisos, deberá hacerse en el módulo, en ir.model.access.csv. Los permisos se activan con valor 1 y se desactivan con valor 0

El sistema modular de Odoo permite que, cada vez que se reinicia el servidor o se actualiza un módulo, se reinterpretan los archivos Python y se leen los XML, actualizando automáticamente todos los cambios

AVISO: Para los módulos de Python hace falta cambiar en odoo.conf la dirección de addons-path a /mnt/extra-addons

UT6: Conceptos Avanzados del Desarrollo de Módulos de Odoo

Relaciones

Generalmente los modelos están relacionados entre ellos. Hay 3 tipos de relaciones en Odoo, las cuales cuadran con los modelos relacionales SQL:

De 1 a Muchos (1,1 en 1:N) -> One2many

De Muchos a 1 (1,n en 1:N) -> Many2one

De Muchos a Muchos -> Many2Many

Relaciones 1:N

Las relaciones 1:N son las que usan One2many y Many2One:

- One2Many es la utilizada mirando desde el lado de la relación que es (1,1)
  Se formula con fields.One2many(moduloContrario, ondelete)
- Many2One es la utilizada mirando desde el lado de la relación que es (1,N)
  Se formula con fields.Many2one(comodel_name, inverse_name)

Relaciones N:N

Las relaciones N:N utilizan Many2many, y representan la relación de “Muchos objetos pueden contener Muchos otros”

- Many2many se formula con fields.Many2many(comodel_name, relacion, columna1 (nombre de la propia columna), columna2 (nombre de la contraria))

Vista Kanban

Las vistas Kanban vienen definidas en etiquetas record, y se utiliza el identificador ir.ui.view. Utilizan la etiqueta <t></t> y siempre llevan atributos que empiezan por t. Se usa <t t-name=””> para elegir una plantilla

Las líneas dentro del modelo Kanban son las siguientes:

- Lo primero que aparecen son los campos del modelo que van a aparecer en el kanban
- Después aparece la etiqueta templates, donde va a empezar todo el “html” de la vista Kanban

<t name=””> para el tipo de template

- sirve para cerar un Kanban con el que se puede interactuar
  - Clases Kanban: Van todos los divs de Kanban, que pueden ser imágenes, detalles o títulos, entre otros
    - o_kanban_details: Campos de registro que se van a ver
      - o_kanban_record_title: Titulo
        - t-if=”record.contenido.value” -> “if exists”
        - t-esc=”record.contenido.value” -> “sout”
      - o_kanban_image: Imagen
        - kanban_image(‘’,’’,record.contenido.rawvalue)

Campos computados

Los campos computados son campos cuyo valor mete el programa solo. Sto se consigue con la propiedad compute = “nombre_funcion”

La función estará creada dentro del propio modelo, con el atributo **self.** Un ejemplo podría ser una función computada para un ID autoincrementado:

```
class prueba(models.Model):
    _id='prueba.prueba1'
    _name='prueba.prueba1'

    id = fields.Char(compute='_aumentar_codigo')

def _aumentar_codigo(self):
    for i in 0..self.length-1:
        self[i].id = i+1
```

UT7: Conceptos Avanzados de Odoo ERP

Botones en las vistas

Las vistas pueden tener botones que hagan alguna acción que defina el usuario. Estos botones se declaran así:

```
<div class="oe_button_box" name="button_box">
    <button name="nombreFuncion" type="object" class="oe_stat_button"
string="textoBoton" icon="icono"/>
</div>
```

type=”object” sirve para ejecutar la función en el objeto asociado a la vista

API: Integración en otros sistemas

Odoo funciona como un servicio web, una aplicación que se encuentra en el lado servidor y que permite que un cliente se conecte a ella para intercambiar información. Tiene de ventaja que se puede acceder desde cualquier lenguaje de programación, además de usar el protocolo HTTP para intercambiar información

Funcionamiento de un servicio Web: Un cliente manda información a un servidor en red y éste hace una petición a la base de datos. La respuesta de la base de datos será enviada al servidor web, quien la transmitirá al cliente.

Servidor REST: Son servicios web que cumplen una serie de requisitos según un patrón de arquitectura y que se ha extendido siendo el patrón predominante. Usan el protocolo HTTP para transferir datos y todas las webs deben tener una URL. La respuesta tiene una estructura determinada (normalmente XML o JSON)

URLs según método:

- Crear – Método POST - /modelo
- Borrar – Método DELETE - /modelo/id
- Buscar – Método GET - /modelo/id
- Buscar varios – Método GET - /modelo
- Actualizar – Método PUT - /modelo/id

Esas URL definen la API del servicio web. La Web API es una parte de los servicios web que permiten usar un único punto de entrada para comunicarse con los servicios web. De esta forma, el servicio web define la lógica del negocio y los clientes solo tienen que buscar según la URL que defina su solicitud.

Función de Solicitud HTTP en Controller:

```
from odoo import http
from odoo.http import Response
import json
class modelo_controller(http.Controller):

  @http.route('/api/modelo', auth='public', method=['GET'], csrf=False)
  def get_modelos(self, **kw):
    try:
      modelos = http.request.env['modulo.modelo'].sudo().search_read([], ['valor1', 'valor2', 'valor3'])
      res = json.dumps(modelos, ensure_ascii=False).encode('utf-8')
      return Response(res, content_type='application/json;charset=utf-8', status=200)
    except Exception as e:
      return Response(json.dumps({'error': str(e)}), content_type='application/json;charset=utf-8',
status=505)
```

Donde:

- @http.route: Establece la ruta del HTTP
  - /api/modelo: Dirección para acceder a la API
  - auth=’public’ -> Indica que el acceso es público (no requiere iniciar sesión)
  - method=[‘GET’] -> Solo permite GET
  - csrf=False -> Desactiva csrf: Una cadena única para confirmar si el usuario que quiere coger los datos es uno verificado.
- Def get_modelos(self, **kw) -> **kw permite recibir tantos parámetros como quiera
  - try
    - http.request.env[‘modulo.modelo’] accede al modelo
    - .sudo() accede en modo administrador
    - search_read([], [valores]) busca todos los objetos del modelo y devuelve esos valores
    - json.dumps(objeto, ensure_ascii=False).encode(utf-8) -> Convierte la lista de registros en un JSON y corrige los caracteres ASCII
    - return Response (respuesta, content_type=’application/json;charset=utf-8’, status=200) Devuelve una respuesta positiva (status code 200) y le pasa como contenido el JSON generado
  - excepción
    - return Response(json.dumps({‘error’: str(e)}), content_type=’application/json;charset=utf-8’, status=505) devuelve un error interno

El resto del Tema aparece explicado en el PDF de Apuntes de Odoo