Estructura Técnica de Odoo

Odoo es una plataforma modular y altamente personalizable que permite desarrollar aplicaciones empresariales integradas. A continuación, se presenta una guía detallada de la estructura técnica de las versiones 15 y 18.

---

1. Estructura Modular de Odoo

Odoo organiza su funcionalidad en **módulos**. Cada módulo se encuentra en un directorio independiente y representa una aplicación o funcionalidad específica. Los módulos pueden ser instalados, desinstalados o actualizados de manera independiente.

- **Ubicación de los módulos:**
    - Odoo 15: /usr/lib/python3/dist-packages/odoo/addons y directorios personalizados especificados en addons_path.
    - Odoo 18: addons_path configurado en el archivo de configuración (odoo.conf).

Cada módulo contiene:

- **__init__.py:** Inicializa el paquete y define las importaciones.
- **__manifest__.py:** Archivo principal del módulo donde se definen los metadatos (nombre, versión, dependencias, etc.).
- **Directorios clave:**
    - models: Define los modelos de datos.
    - views: Contiene las vistas XML para la interfaz.
    - security: Define las reglas de acceso y grupos de usuarios.
    - data: Archivos de datos iniciales como secuencias o registros predeterminados.
    - static: Recursos estáticos como CSS, JavaScript e imágenes.

---

2. Modelos en Odoo (ORM)

El ORM (Object Relational Mapping) de Odoo simplifica la interacción con la base de datos. Los modelos en Odoo se definen como clases de Python que heredan de models.Model.

- **Estructura Básica:**

```
from odoo import models, fields, api


class ExampleModel(models.Model):
    _name = 'example.model'
    _description = 'Example Model'

    name = fields.Char(string='Name', required=True)
    description = fields.Text(string='Description')
    active = fields.Boolean(string='Active', default=True)
```

- **Campos Comunes:**
    - fields.Char: Cadenas de texto.
    - fields.Text: Texto largo.
    - fields.Integer: Enteros.
    - fields.Many2one: Relación muchos a uno.
    - fields.One2many: Relación uno a muchos.
    - fields.Many2many: Relación muchos a muchos.
- **Parámetros Importantes:**
    - _name: Nombre técnico del modelo.
    - _description: Descripción del modelo.
    - _inherit: Herencia de modelos existentes.
    - _order: Ordenamiento predeterminado.
- **Herencia de Modelos:**
    - **Clásica:** Extiende un modelo existente.

- class InheritedModel(models.Model):
- _inherit = 'existing.model'
-

new_field = fields.Char(string='New Field')

    - **Delegada:** Combina varios modelos mediante una relación Many2one.

---

3. Vistas y Arquitectura de la Interfaz

Las vistas en Odoo definen la estructura de la interfaz de usuario (UI). Se configuran mediante archivos XML.

- **Tipos de Vistas:**
    - **Formulario (form):** Para crear o editar registros.
    - **Lista (tree):** Presenta registros en forma de tabla.
    - **Kanban:** Vista visual para procesos.
    - **Gráfico:** Reportes gráficos.
    - **Calendario:** Visualiza datos en un calendario.
- **Ejemplo de Definición de Vista Formulario:**

```
<record id="view_example_form" model="ir.ui.view">
  <field name="name">example.form.view</field>
  <field name="model">example.model</field>
  <field name="arch" type="xml">
    <form>
      <sheet>
        <group>
          <field name="name"/>
          <field name="description"/>
        </group>
```

```
        </sheet>
      </form>
    </field>
</record>
```

---

4. Seguridad y Acceso

La seguridad en Odoo se gestiona a través de reglas y grupos de usuarios:

- **Grupos de Usuarios:** Definidos en security/ir.model.access.csv y en XML.
- **Reglas de Acceso:** Definen los permisos para leer, escribir, crear o eliminar registros.
- **Ejemplo de Archivo ir.model.access.csv:**

```
id,name,model_id:id,group_id:id,perm_read,perm_write,perm_create,perm_unlink
access_example_model,access_example_model,model_example_model,base.group_user,1,1,1,0
```

- **Ejemplo de Regla de Seguridad:**

```
<record id="example_rule" model="ir.rule">
    <field name="name">Example Rule</field>
    <field name="model_id" ref="model_example_model"/>
    <field name="domain_force">['|', ('user_id', '=', user.id), ('public', '=', True)]</field>
</record>
```

---

5. QWeb: Plantillas y Reportes

Odoo utiliza QWeb para definir vistas y reportes PDF. Las plantillas QWeb están escritas en XML y permiten la generación dinámica de contenido.

- **Ejemplo de Plantilla QWeb:**

```
<template id="report_example">
    <t t-call="web.external_layout">
```

```
      <div class="page">
        <h2>Example Report</h2>
        <t t-foreach="docs" t-as="doc">
          <p><t t-esc="doc.name"/></p>
        </t>
      </div>
    </t>
</template>
```

- **Vinculación de Reporte:**

```
<record id="action_report_example" model="ir.actions.report">
    <field name="name">Example Report</field>
    <field name="model">example.model</field>
    <field name="report_type">qweb-pdf</field>
    <field name="report_name">module_name.report_example</field>
</record>
```

---

6. Actualizaciones de la Versión 18

Odoo 18 introduce cambios significativos en comparación con la versión 15:

- Cambios en las vistas de tipo “list”, reemplazando “tree”.
- Mejoras en la gestión de eventos asincrónicos y soporte a interfaces responsivas.
- Mejoras de rendimiento en el ORM y soporte extendido para PostgreSQL 15+.

---

7. Integración de Módulos Personalizados

Para integrar módulos personalizados en Odoo:

1. Crear el módulo siguiendo la estructura estándar.

2. Configurar el archivo __manifest__.py con las dependencias necesarias.
3. Colocar el módulo en el directorio especificado por addons_path.
4. Actualizar la lista de módulos con:

odoo -u your_module_name -d your_database_name

Esta guía proporciona una base para entender y trabajar con la estructura técnica de Odoo. Si necesitas más detalles sobre un tema específico, no dudes en mencionarlo.