INTEGRACIÓN DE SISTEMAS EMPRESARIALES AVANZADO

LABORATORIO N° 10

Consumo de API Externa

Alumno(s):		***Nota***	
Grupo:		***Ciclo: VI***	

Criterio de Evaluación	*Excelente (4pts)*	*Bueno (3pts)*	*Requiere mejora (2pts)*	*No acept. (0pts)*	*Puntaje Logrado*
Identificar métodos de la API de modelos de Odoo					
Consumir APIs Externas					
Configura un modelo desde el inicio					
Documenta los acuerdos y detalles del trabajo a realizar					
Es puntual y redacta el informe adecuadamente					

Laboratorio 10: Consumo de API Externa

Objetivos:

Al finalizar el laboratorio el estudiante será capaz de:
- Identificar los principales requerimientos del trabajo a realizar
- Definir las responsabilidades de los integrantes del grupo
- Documentar los alcances del trabajo a realizar

Seguridad:

- Ubicar maletines y/o mochilas en el gabinete del aula de Laboratorio.
- No ingresar con líquidos, ni comida al aula de Laboratorio.
- Al culminar la sesión de laboratorio apagar correctamente la computadora y la pantalla, y ordenar las sillas utilizadas.

Equipos y Materiales:

- Una computadora con:
  - Windows 7 o superior
  - Conexión a la red del laboratorio
  - Software de virtualización (Opcional)
  - Software ERP ODOO instalado

Procedimiento:

1. **Configuración inicial.**
   1.1. Proseguiremos modificando el módulo **facturacion** creado en el laboratorio anterior.

2. **Creación de modelo Documentos**
   2.1. Crearemos el archivo documentos.py dentro de la carpeta models con el siguiente contenido:

```python
# -*- coding: utf-8 -*-
from odoo import models, fields, api

class Documentos(models.Model):
    _name = 'facturacion.documentos'
    _description = 'Documentos para las entidades'

    name = fields.Char(string='Nombre del documento')
    code = fields.Char(string='Codigo del documento')
```

   2.2. Al haber creado un nuevo archivo Python, debemos agregarlo en nuestro __init__.py dentro de models

```python
# -*- coding: utf-8 -*-
from . import series
from . import documentos
from . import account_invoice
```

   2.3. Procederemos a la creación de la vista correspondiente. Crearemos el archivo documentos_view.xml dentro de la carpeta views con el contenido a continuación.

```xml
<?xml version="1.0" encoding="utf-8"?>
<odoo>
    <data>
        <record id="documentos_tree_view" model="ir.ui.view">
            <field name="name">documentos.tree.view</field>
            <field name="model">facturacion.documentos</field>
            <field name="arch" type="xml">
                <tree string="Documentos" editable="bottom">
                    <field name="name"/>
                    <field name="code"/>
                </tree>
            </field>
        </record>

        <record model="ir.actions.act_window" id="facturacion.action_documentos">
            <field name="name">Listado de documentos</field>
            <field name="res_model">facturacion.documentos</field>
            <field name="view_mode">tree,form</field>
        </record>

        <menuitem name="Documentos" id="facturacion.menu_1_documentos"
            parent="facturacion.menu_facturacion" action="facturacion.action_documentos"/>
    </data>
</odoo>
```

Note que a diferencia de las series, no estamos creando un formulario o una búsqueda, sino solamente el tree y el menuitem. Además, cabe resaltar el atributo editable=”bottom”. Este permite que el listado mismo sea editable sin cambiar de vista.

2.4. Agregaremos dentro de la carpeta data un archivo llamado documentos.xml con el siguiente contenido

```xml
<odoo>
    <data>
        <record id="documento1" model="facturacion.documentos">
            <field name="name">Doc.trib.no.dom.sun.ruc</field>
            <field name="code">0</field>
        </record>
        <record id="documento2" model="facturacion.documentos">
            <field name="name">Doc. Nacional de identidad</field>
            <field name="code">1</field>
        </record>
        <record id="documento3" model="facturacion.documentos">
            <field name="name">Carnet de extranjeria</field>
            <field name="code">4</field>
        </record>
        <record id="documento4" model="facturacion.documentos">
            <field name="name">Registro Unico de Contribuyentes</field>
            <field name="code">6</field>
        </record>
        <record id="documento5" model="facturacion.documentos">
            <field name="name">Pasaporte</field>
            <field name="code">7</field>
        </record>
        <record id="documento6" model="facturacion.documentos">
            <field name="name">Ced.Diplomatica de Identidad</field>
            <field name="code">A</field>
        </record>
```

```
        <record id="documento7" model="facturacion.documentos">
            <field name="name">Documento identidad pais residencia-no.d</field>
            <field name="code">B</field>
        </record>
        <record id="documento8" model="facturacion.documentos">
            <field name="name">Tax Identification Number - TIN - Doc Trib PP.NN</field>
            <field name="code">C</field>
        </record>
        <record id="documento9" model="facturacion.documentos">
            <field name="name">Identification Number - IN - Doc Trib PP.JJ</field>
            <field name="code">D</field>
        </record>
        <record id="documento10" model="facturacion.documentos">
            <field name="name">TAM - Tarjeta Andina de Migracion</field>
            <field name="code">E</field>
        </record>
    </data>
</odoo>
```

2.5. Para finalizar, agregaremos la dependencia contacts (ya que luego modificaremos a los contactos del sistema) y los archivos de vista documentos.xml y documentos_view.xml dentro del __manifest__.py

```
# -*- coding: utf-8 -*-
{
    'name' : 'Facturacion',
    'version' : '1.0',
    'summary': 'Modulo basico para facturacion en Peru',
    'description': """
Core mechanisms for the accounting modules. To display the
    """,
    'depends' : ['account','contacts'],
    'data': [
            'data/series.xml',
            'data/documentos.xml',
            'views/series_view.xml',
            'views/documentos_view.xml',
            'views/account_invoice_view.xml',
    ]
}
```

Reiniciaremos el servicio y luego actualizaremos el módulo. Adjunte un GIF del funcionamiento de este nuevo menú Documentos, mostrando como se edita un ítem sin necesidad de abrir un formulario.

3. Relación con contactos

3.1. Crearemos el archivo res_partner.py dentro de la carpeta models con el siguiente contenido.

```
# -*- coding: utf-8 -*-
from odoo import models, fields, api

class ResPartner(models.Model):
    _inherit = 'res.partner'

    tipo_doc_id = fields.Many2one(comodel_name='facturacion.documentos',
        string='Tipo de documento')
```

Lo que estamos haciendo es referenciar al modelo antes creado para saber qué tipo de documento tiene la entidad. Nos falta agregar dicho campo a las vistas.

3.2. Debemos agregar res_partner dentro del __init__.py de la carpeta models.

```
# -*- coding: utf-8 -*-
from . import series
from . import documentos
from . import account_invoice
from . import res_partner
```

3.3. Así mismo, crearemos el archivo res_partner_view.xml dentro de la carpeta views con el siguiente contenido.

```
<?xml version="1.0" encoding="UTF-8"?>
<odoo>
    <data>
        <record model='ir.ui.view' id='view_partner_form'>
            <field name='name'>res.partner.form</field>
            <field name='model'>res.partner</field>
            <field name='inherit_id' ref='base.view_partner_form'/>
            <field name='arch' type='xml'>
                <field name="vat" position="before">
                    <field name="tipo_doc_id"/>
                </field>
                <field name="vat" position="attributes">
                    <attribute name="string">RUC/DNI</attribute>
                </field>
            </field>
        </record>
    </data>
</odoo>
```

Cabe resaltar como se está realizando dos cosas al mismo tiempo:

- Estamos agregando el campo tipo_doc_id antes del vat (Documento de la Entidad)
- Estamos modificando el atributo string del VAT, para que ahora diga RUC/DNI

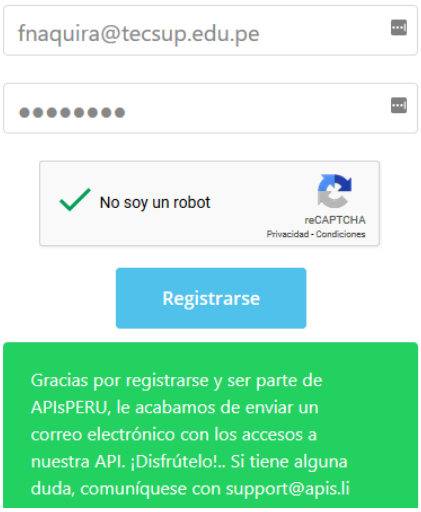
3.4. Debemos agregar dicha vista al __manifest__.py

```
    'depends' : ['account','contacts'],
    'data': [
        'data/series.xml',
        'data/documentos.xml',
        'views/series_view.xml',
        'views/documentos_view.xml',
        'views/account_invoice_view.xml',
        'views/res_partner_view.xml',
    ]
}
```

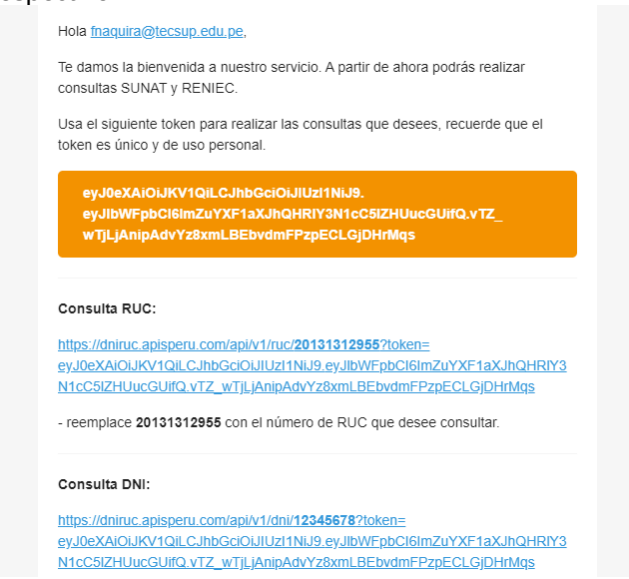
Reiniciaremos el servicio y luego actualizaremos el módulo. Adjunte una captura de cómo se visualiza el nuevo campo al momento de crear un contacto.

4. Consumo de API externa
   4.1. Ahora que tenemos el tipo de documento, haremos que el sistema complete los datos según si se escoge DNI o RUC. Ingresemos a la siguiente URL: https://www.apisperu.com/

   4.2. En la parte inferior encontraremos un formulario para registrarnos. Utilice su correo de Tecsup para utilizar este servicio de validación de documentos.



   4.3. En nuestra bandeja, tendremos un correo de dicho servicio con un token. Conservaremos este token para nuestro consumo respectivo.



4.4. Vamos a proceder a modificar nuestro res_partner.py. Agregaremos las siguientes librerías:

- requests (Sirve para realizar consultas tipo GET, POST, etc)
- json (sirve para transformar objetos JSON en diccionarios. Necesitaremos esto para las respuestas de la API)
- logging (La usaremos con fines didácticos, para mostrar lo que sucede en el sistema, ya que es una librería de impresión en consola)

Así mismo, declararemos dos variables globales, URL y TOKEN. En el caso de Token, coloque la recibida en el correo del servicio.

```
# -*- coding: utf-8 -*-
from odoo import models, fields, api
from odoo.exceptions import Warning
import requests
import json
import logging
log = logging.getLogger(__name__)

URL = 'https://dniruc.apisperu.com/api/v1/'
TOKEN = 'eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJlbWFpbCI6ImZuYXF1aXJhQHRlY3N1cC5lZHUucGUifQ.vTZ_wTjLjAnipAdvYz8xmLBEbvdmFPzpECLGjDHrMqs'

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

4.5. Ahora, agregaremos un método onchange. Los onchange son métodos de la API de Odoo que escuchan a los campos con los que se les declaran, y cada vez que tienen un cambio ejecutan dicha función. Agregue al final del archivo (tenga cuidado con la identación, debe tener al menos un bloque identado) el siguiente código, que escuchará a cualquier cambio de tipo de documento o del documento mismo.

```
    @api.multi
    @api.onchange('vat','tipo_doc_id')
    def onchange_doc(self):
        for partner in self:
            log.info('----------------')
            log.info('Inicia nuestra validacion')
            log.info(partner.vat)
            log.info(partner.tipo_doc_id)
```

Reinicie el servicio. Ahora, intente crear o editar un contacto, luego adjunte una captura de lo reflejado en el log de Odoo.

4.6. Seguiremos modificando nuestra función. Ya que la API a consumir solamente devuelve los valores de DNI o RUC, preguntaremos en una condición IF si es que el tipo de documento está entre ambos valores, y al mismo tiempo, si el vat (o documento del contacto) no está vacío. Aprovecharemos también para determinar si válido cuando es de tipo DNI (cuando tiene ocho dígitos) y si es válido cuando es de tipo RUC (cuando tiene once dígitos)

```
        for partner in self:
            log.info('----------------')
            log.info('Inicia nuestra validacion')
            log.info(partner.vat)
            log.info(partner.tipo_doc_id)
            if partner.tipo_doc_id.code in ('1','6') and partner.vat is not False:
                tipo_doc = False
                if partner.tipo_doc_id.code == '1' and len(partner.vat) == 8:
                    tipo_doc = 'dni'
                if partner.tipo_doc_id.code == '6' and len(partner.vat) == 11:
                    tipo_doc = 'ruc'
```

4.7. Luego de haber hecho esta comprobación, y tener la certeza de que tenemos la data necesaria, realizaremos la consulta a la API. Agregaremos las siguientes líneas que invocarán a la función get_doc.

```
            if tipo_doc:
                respuesta = self.get_doc(tipo_doc,partner.vat)
                if not respuesta:
                    raise Warning("Server not responding now, try again later")
                if not respuesta[0]:
                    raise Warning(respuesta[1])
```

Sin embargo, la función get_doc no existe. Agregaremos la siguiente porción de código líneas abajo o líneas arriba. Para que Python sepa que es una función distinta, debemos cuidar que la identación sea correcta (fíjese que este pedazo de código está identado solamente un bloque).

```
    def get_doc(self,tipo_doc,vat):
        s = requests.Session()
        try:
            r = s.get(URL+tipo_doc+'/'+vat+'?token='+TOKEN, verify=False)
        except requests.exceptions.RequestException as e:
            return (False, e)
        if r.status_code == 200:
            return (True, r.content)
        else:
            return (False, 'Tenemos un problema!')
```

Dentro de esta función, estamos invocando a la librería requests, creando una sesión de consulta tipo GET, a la que le pasaremos la URL compuesta (concatemos la URL principal, más el tipo de documento, más el documento, más el token). Así mismo, en este caso especial, agregaremos el flag verify=False, debido a que la gente de APIS PERU utiliza un certificado digital firmado por ellos mismos. Si intenta ejecutar este código sin dicho flag, el servidor arrojaría un error de Problemas de seguridad con el Certificado del Servidor.

4.8. Volvemos a nuestro método onchange. Después de invocar a get_doc, agregamos el siguiente código:

```
        if tipo_doc:
            respuesta = self.get_doc(tipo_doc,partner.vat)
            if not respuesta:
                raise Warning("Server not responding now, try again later")
            if not respuesta[0]:
                raise Warning(respuesta[1])
            try:
                respuesta_json = json.loads(respuesta[1].decode())
            except ValueError as e:
                raise Warning(("Server response content is not serializable to JSON object: %s" % e))
            if tipo_doc == 'ruc':
                partner.name = respuesta_json['razonSocial']
                partner.street = respuesta_json['direccion']
            if tipo_doc == 'dni':
                partner.name = respuesta_json['nombres'] + ' ' + respuesta_json['apellidoPaterno'] + ' ' + respuesta_json['apellidoMaterno']
```

Como se puede apreciar, en caso se obtenga una respuesta válida del servidor, la traduciremos con json.loads, que convierte un JSON en una variable entendible por Python (lo que solemos llamar un diccionario).
Finalmente, en caso sea RUC o DNI, basta con invocar al objeto partner (nuestro contacto) y decirle que campos queremos modificar, en el caso de RUC, modificamos name y street, ya que esa data nos entrega la API.

4.9. Reinicie el servidor y actualice el módulo.
Adjunte un GIF mostrando como selecciona DNI e ingresa su DNI y hacer click fuera del campo, Odoo autocompleta su Nombre.
Adjunte un GIF mostrando como selecciona RUC e ingresa el RUC de la empresa Saga Falabella (debe googlear esta información) y lo que autocompleta Odoo.

5. Constraint o limitantes.
   5.1. Odoo también provee un método llamado constrains que restringe la creación de un contacto o su edición, en caso cumpla una condición. Por ejemplo, nosotros limitaremos la creación de dos contactos con el mismo documento.
   5.2. Modificaremos el archivo res_partner.py para que tenga el siguiente nuevo método.

```
@api.constrains('tipo_doc_id','vat')
def _check_unique_vat(self):
    for partner in self:
        if not partner.vat:
            return True
        same_vat_count = self.search_count([
            ('tipo_doc_id','=',partner.tipo_doc_id.id),
            ('vat','=',partner.vat)
        ])
        if same_vat_count > 1:
            raise Warning('Ya existe una entidad con ese documento!')
```

En caso de que el vat este vacío, no tendremos ningún problema. Pero en caso contrario, haremos una búsqueda de cuantos registros existen con dicho documento y tipo de documento. Para eso utilizaremos el método search_count y le pasaremos nuestras condiciones de búsqueda.
Las condiciones de búsqueda (o más conocidas como dominios) se suelen escribir en tres partes, haciendo referencia al campo a comparar, la operación (en este caso, = de comparación, pero puede ser < menor, > mayor, in de inclusión, not in de exclusión, etc.) para más detalles, puede consultarse la sección https://www.odoo.com/documentation/11.0/reference/orm.html#domains
En caso, encontremos más de un registro con dicho documento, arrojanremos una advertencia al usuario.

   5.3. Reinicie el servidor y actualice el módulo.
   Adjunte un GIF mostrando dicha advertencia al intentar crear otra vez un contacto con su mismo DNI.

6. Campos relacionados
   6.1. En algunas ocasiones, necesitaremos acceder a data relacionada a un campo many2one. Por ejemplo, en el caso de las facturas, si bien hemos seleccionado la serie, no sabemos que tipo de comprobante es. Esto si está en cambio dentro de la serie.

   6.2. Modificaremos el archivo account_invoice.py para agregar el campo tipo_doc.

```
class AccountInvoice(models.Model):
    _inherit = 'account.invoice'

    serie_id = fields.Many2one(comodel_name='facturacion.series',
        string='Serie Electrónica')
    tipo_doc = fields.Selection(string='Tipo de Documento',related='serie_id.document_type',readonly=True)
```

Este campo tiene el flag related, que hace referencia al campo serie_id y luego a su subcampo (es decir, el tipo de documento). Los campos related deben ser del mismo tipo al que hacen referencia, en este caso, como hacemos referencia a un selection, debemos declararlo igual.
Asi mismo, agregaremos el flag readonly en True, ya que no queremos editar un campo que está relacionado a la serie que escojamos.

6.3. Agregaremos dentro del archivo account_invoice_view.xml la línea de tipo_doc para mostrar dicha relación.

```xml
<?xml version="1.0" encoding="utf-8"?>
<odoo>
    <data>
        <record id="invoice_form_series" model="ir.ui.view">
            <field name="name">invoice.series.form.view</field>
            <field name="model">account.invoice</field>
            <field name="inherit_id" ref="account.invoice_form"/>
            <field name="arch" type="xml">
                <field name="date_invoice" position="after">
                    <field name="serie_id"/>
                    <field name="tipo_doc"/>
                </field>
            </field>
        </record>
    </data>
</odoo>      You, 7 days ago • se añade series por defecto
```

6.4. Reinicie el servidor y actualice el módulo.
Adjunte un GIF mostrando el comportamiento de nuestro nuevo campo cuando, al crear una factura de Cliente, seleccionamos una serie de boleta o una serie de facturas.