PRACTICA 13
OBJETIVO: Acciones e Informes Personalizados
DESCRIPCIÓN:

En el tema anterior estudiamos la generación de informes a través del elemento report que proporciona Odoo, simplificando mucho esta tarea.
En este tema vamos a profundizar en la generación de los informes, en concreto la generación de informes personalizados.
Además, profundizaremos nuestros conocimientos para crear una entrada en el menú que dispare la acción de generar un informe.

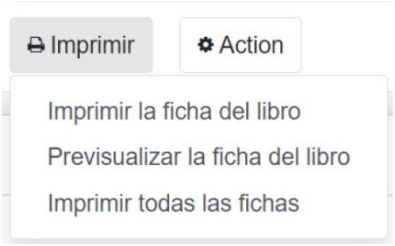
ACCIONES:
Las acciones definen el comportamiento de Odoo en respuesta a la interacción de los usuarios, como, por ejemplo, cuando un usuario inicia sesión, hace clic en un botón, selecciona una factura,
Las acciones se pueden almacenar en la base de datos o, mediante la sentencia return de los métodos, devolverse como diccionarios de Python para que se ejecuten
Todas las acciones comparten dos atributos obligatorios:

- **type** especifica la categoría de la acción, cada categoría indica cómo se interpreta la acción y los campos que requiere para poderse usar
- **name** contiene una b reve descripción legible por el usuario de la acción, puede mostrarse en la interfaz del cliente

Se pueden consultar todas las categorías de acciones de Odoo, los tipos de acciones en
https::://www.odoo.com/documentation/15.0/es/developer/reference/backend/actions.html

Además de los dos atributos obligatorios, todas las acciones disponen de atributos opcionales que se utilizan para presentar una acción en el menú contextual de un modelo arbitrario:

- **binding_model_id** especifica a qué modelo se vincula la acción Para acciones del servidor se usa model_id.
- **binding_type** indica el tipo de vinculación, que es principalmente en qué menú contextual aparecerá la acción.
  - **action** si no se especifica indica que la acción aparecerá en el menú contextual Action del modelo al que está vinculada la acción.
  - **report** especifica que la acción aparecerá en el menú contextual Imprimir del modelo al que está vinculada la acción



- **binding_view_types** contiene una lista separada por comas de los tipos de vista para los que aparece la acción en el menú contextual El valor predeterminado es list,form que se corresponden con la vista árbol y la vista formulario.

Por último, de todas las categorías de acción que existen, ya hemos vista algunas en las prácticas anteriores cuando creábamos vistas, menús debemos estudiar la que dispara la generación de informes, que es ir actions report

PASO 1: ir.actions.report

La forma de generar informes personalizados es creando una acción ir actions report mediante la etiqueta record que generará un registro en la tabla de ir_act_report_xml de la base de datos
Los parámetros de la acción ir actions report a parte de los que se acaban de presentar en las diapositivas anteriores (los obligatorios y los opcionales a todas las acciones), son muy parecidos a los del elemento report que se estudio en el tema anterior

- **report_type** (por defecto qweb pdf con el valor qweb pdf genera informes en PDF y con el valor qweb html informes en formato HTML
- **print_report_name** contiene una expresión de Python que define el nombre del fichero PDF attachment_use si se establece con valor True el informe se almacenará como un archivo adjunto del registro, utilizando el nombre generado por la expresión attachment Este campo permite generarun informe una única vez, lo que puede ser útil si motivos legales, como por ejemplo, la firma de un contrato, un comprobante de pago,...
- **attachment** contiene una expresión de Python que define el nombre del informe El registro, junto con los datos del mismo, es accesible a través de la variable object

- **paperformat** especifica el id externo del formato de papel que se desea usar para el informe Si no se especifica, se utiliza el formato predeterminado de papel de la empresa

Además es obligatorio y muy necesario el campo:

- **report_name** que especifica el nombre del id externo de la plantilla que se va a usar, y el modelo abstracto necesario que permite, a través de su método get_report_values seleccionar los registros para generar el informe

PASO 2: _get_report_values

Cuando se dispara una acción de impresión el sistema de generación de informes debe acceder y comprender los campos de los registros del modelo que se va a usar para generar el informe
El id de ese modelo se especificó al definir el informe, mediante el campo model de la etiqueta record
como por ejemplo <field name=”model”> biblioteca libro </field>
Odoo primero busca un modelo llamado como report module report_name e invoca al método get_report_values doc_ids data) de dicho modelo para preparar los datos de representación
para la plantilla
Así, si el report_name es biblioteca report_imprimir_fichas especificado mediante el campo report_name de la etiqueta record el método a invocar es
report.biblioteca report_imprimir_fichas get_report_values doc_ids data)

Por lo que es necesario crear un modelo llamado report biblioteca report_imprimir_fichas que contenga el método get_report_values self docids None data= None
El modelo es de tipo models AbstractModel ya que este modelo abstracto, no contiene registros, esto es, no van a existir instancias de dicha clase, porque no va a almacenar registros en la base de datos,
solo va a contener el método para crear el contexto de impresión Se recomienda repasar los contenidos de la práctica **MODELO SIMPLE**
El método get_report_values self docids None data= None recibe como parámetros self los id de los docs de los registros seleccionados mediante docids y data
El parámetro data contiene el contexto El contexto en Odoo es parte del entorno de un conjunto de registros Básicamente, el contexto es un diccionario de Python que contiene los datos de la sesión, pero lo más importante es que se puede usar el contexto para pasar los datos específicos a una método en Odoo

En relación con el método get_report_values a través del parámetro data se pasan datos como el idioma que está configurado, la zona horaria, datos de sesión el usuario que ha lanzado la impresión y a
que compañía pertenece, y el tipo de informe, PDF o HTML
Por ejemplo:

```
{' context ': lang ': es_ES ', tz ': Europe /Madrid', uid ': 2,
allowed_company_ids ': [1]}, report_type ': pdf}
```

Dentro del cuerpo del método get_report_values se pueden seleccionar los registros que desean utilizarse para generar un informe
A veces, solo se quiere generar un informe con los registros que cumplen una serie de condiciones, como por ejemplo, las facturas que pertenezcan al mes anterior, los pagos a un determinado cliente, etc.
Como vimos en el tema 10 estas condiciones se especifican mediante dominios

PASO 3: dominios

Un dominio de Odoo se utiliza para seleccionar registros de un modelo
Es un caso de uso muy común cuando necesita mostrar un subconjunto de todos los registros disponibles de una acción, o permitir que solo un subconjunto de posibles registros sea el objetivo de una relación *Many2One*.
La forma de describir estos filtros en Odoo se llama dominio Se puede definir como una lista de operaciones que se utilizan para filtrar sus datos o para realizar búsquedas
Cada condición de un dominio contiene un nombre de campo, un operador y un valor .
La sintaxis es la siguiente:

```
Qdominio="[(nombre_del_campo , 'operador', 'valor')]"
```

nombre_del_campo es el nombre de un campo del modelo

El operador puede ser cualquiera de los que se muestran a continuación:

- De comparación: <,>, =, !=, <=, >=
- like este operador coincide con el patrón que contenga el valor buscado, , especificado en valor. Distingue entre mayúsculas y minúsculas. Por ejemplo: [('nombre', like ', 'os')] valdría para oscar , jose ,…y no para óscar o Jóse
- =like se utiliza para comparar con una patrón exacto, especificado en valor, pero pueden usarse comodines de caracteres como un guion bajo en el patrón representa a cualquier carácter individual, un signo de porcentaje representa a una cadena de cero o más caracteres Distingue entre mayúsculas y minúsculas Por ejemplo ::[(' like odoo valdría para podoo ,,[(' like odoo valdría para podoo odoo Tytysydystodoo...
- not like como el operador like pero en este caso que no coincida con el patrón, especificado en valor Distingue entre mayúsculas y minúsculas.

El operador puede ser cualquiera de los que se muestran a continuación:

- **ilike** operador igual que like pero no distingue entre mayúsculas y minúsculas.
- **=ilike** operador igual que like pero no distingue entre mayúsculas y minúsculas
- **not ilike** como el operador not like pero no distingue entre mayúsculas y minúsculas
- **in** es igual a cualquiera de los elementos especificados en valor Valor debe ser especificado como una lista de elementos.
- **not in** no es igual a cualquiera de los elementos especificados en valor Valor debe ser especificado como una lista de elementos.
- **child_of** es un registro hijo (de un valor Tiene en cuenta la semántica del modelo, por ejemplo, siguiendo el campo de relación llamado parent_name.

Se pueden utilizar los siguientes operadores para combinar elementos del dominio:

- **AND** lógico, por defecto
- **OR** lógico
- **NOT** lógico

Por ejemplo, para buscar patrones de nombre ABC , de Bélgica o Alemania, cuyo idioma no sea el inglés:

```
[('nombre','=','ABC'),
('language.code en_US '|',
(' country_id.code ','=',' ('country_id.code ','=','de')]
```

equivalente en seudocódigo a:

(el nombre es 'ABC')
Y (el idioma NO es inglés)
Y (el país es Bélgica O Alemania)

PASO 4: Informes personalizados

Recapitulando lo que hemos visto hasta ahora, para crear un informe personalizado se debe crear un registro record con los parámetros que define la acción de tipo **ir.actions.report**
Estos parámetros permiten especificar y definir cómo se creará el informe el informe, como son el modelo de donde se sacan los datos a imprimir, el texto que debe aparecer en los menús contextuales
Este record (será procesado cuando se produzca la acción para componer el informe
Además, como con el elemento report se necesita una plantilla que genere el contenido y el aspecto del informe
Por último, sin la facilidad que aportaba el elemento report también es necesario crear un método, llamado ***_get_report_values*** que es invocado cuando se produce la acción de imprimir y es el encargado de seleccionar, mediante una expresión de dominio, los registros que se desean imprimir y que se pasarán a la plantilla.

De igual modo, también está disponible un modelo de plantilla para definir el cuerpo del informe, aunque puede crearse desde cero una nueva plantilla para informes personalizados, que permita, por ejemplo, expandir la funcionalidad, dando acceso a más modelos,
Y los pasos necesarios para crear un informe personalizado son:

- Se define una nueva acción del tipo ir.actions.report a través de la etiqueta <record>
- Se define una nueva plantilla, mediante el uso de una plantilla ya predefinida en Odoo, que generará el contenido y la forma del informe.
- Se crea una clase abstracta con un método llamado _get_report_values, que permite seleccionar aquellos datos que se quieren mostrar en el informe y que serán proporcionados a la plantilla

Vamos a crear un informe personalizado que genere en un informe todas las fichas de los libros del ejemplo de los temas anteriores
Lo primero que hacemos es crear una acción ir.actions.report que representa a este nuevo informe
Para ello editamos el fichero reports /informes xml y añadimos el siguiente código:

```
<record id="action_report_imprimir_fichas" model="ir.actions.report">
        <field name="name">Imprimir todas las fichas</field>
        <field name="model">biblioteca.libro</field>
        <field name="report_type">qweb-pdf</field>
        <field name="report_name">biblioteca.report_imprimir_fichas</field>
        <field name="binding_model_id" ref="model_biblioteca_libro"/>
        <field name="binding_type">report</field>
</record>
```

a) Ejemplo:

El código anterior crea una acción de tipo ***ir.actions.report*** con el id *action_report_imprimir_fichas*.

Con <field name="name">Imprimir todas las fichas</field> indicamos que queremos que aparezca el texto Imprimir todas las fichas en el menú contextual Imprimir (<field name="binding_type">report</field>) del modelo biblioteca.libro (<field name="binding_model_id" ref="model_biblioteca_libro"/>)

Cuando el usuario haga clic en imprimir se lanzará una acción imprimir que generará un informe en PDF (<field name="report_type">qweb-pdf</field>) y que usará la plantilla biblioteca.report_imprimir_fichas (<field name="report_name">biblioteca.report_imprimir_fichas</field>) a partir de los registros del modelo biblioteca.libro (<field name="model">biblioteca.libro</field>)

Lo siguiente es crear la plantilla biblioteca.report_imprimir_fichas que usa el informe Imprimir todas las fichas.
La plantilla la crearemos dentro del fichero views/templates.xml, que es el utilizado para plantillas.
La plantilla que queremos crear tiene la siguiente estructura:
Sin embargo, podemos observar que es repetir la plantilla report_ficha_libro que ya creamos en el tema anterior, solo que está ocasión, en vez de pasarle un único registro, le pasaremos todos los registro para que genere una ficha para cada uno.
Así que vamos a reutilizar la plantilla ya creada.

```
<template id="report_imprimir_fichas">
        <t t-call="web.html_container">
        ...
        </t>
</template>
```

Crearemos una plantilla que generará el cuerpo del informe, y que usará tanto report_ficha_libro como report_imprimir_fichas.

b) Creamos la plantilla:

```
<template id="plantilla_ficha_libro">
</template>
```

Dentro del cuerpo de esta plantilla, metemos el código de la plantilla report_ficha_libro desde la etiqueta **<t t-foreach="docs" t-as="o">** hasta su etiqueta de cierre </t> correspondiente.

Modificamos la plantilla report_ficha_libro, borrando el código que hemos colocado en la plantilla plantilla_ficha_libro y hacemos que llame a la dicha plantilla para que utilice su código mediante la etiqueta: **<t t-call="biblioteca.plantilla_ficha_libro"></t>.**
Como queremos lo mismo para la plantilla report_imprimir_fichas, hacemos una copia de la plantilla report_ficha_libro y cambiamos su id por report_imprimir_fichas.

c) Al final, la plantilla report_ficha_libro queda de la siguiente forma:

```
<template id="report_ficha_libro">
	<t t-call="web.html_container">
		<t t-call="biblioteca.plantilla_ficha_libro"></t>
		</t>
</template>
```

Y la plantilla report_ficha_libro:

```
<template id="report_imprimir_fichas">
	<t t-call="web.html_container">
		<t t-call="biblioteca.plantilla_ficha_libro">
		</t>
	</t>
</template>
```

d) Ejemplo resultante

Y la plantilla plantilla_ficha_libro:

```
<template id="plantilla_ficha_libro">
    <t t-foreach="docs" t-as="o">
        <t t-call="web.external_layout">
            <div class="page">
                <h2><span t-field="o.name"/></h2>
                <p style="color: black; font-family: Verdana; font-weight: bold;">Título:
                    <span t-field="o.name"/>
                </p>
                <t t-if="o.imagen">
                    <img t-att-src="'%s' % to_text(o.imagen)" style="max-height:8cm; max-width:8cm;"/>
                </t>
                <p></p>
                <t t-if="o.author_ids">
                    <p><span style="color: black; font-family: Verdana; font-weight: bold;">Autor/es:</span>
                    <span t-field="o.author_ids" />
                    </p>
                </t>
                <t t-if="o.date_release">
                    <p><span style="color: black; font-family: Verdana; font-weight: bold;">Fecha de publicación:</span>
                    <span t-field="o.date_release" />
                    </p>
                </t>
                <t t-if="o.language">
                    <p><span style="color: black; font-family: Verdana; font-weight: bold;">Idioma:</span>
                    <span t-field="o.language" />
                    </p>
                </t>
                <t t-if="o.pages">
                    <p><span style="color: black; font-family: Verdana; font-weight: bold;">Páginas:</span>
                    <span t-field="o.pages" />
                    </p>
                </t>
                <t t-if="o.summary">
                    <p><span style="color: black; font-family: Verdana; font-weight: bold;">Resumen:</span>
                    <span t-field="o.summary" />
                    </p>
                </t>
            </div>
        </t>
    </t>
</template>
```

e) Añadir el método _get_report_value

Solo falta añadir el método _get_report_values que seleccionará los registros del modelo biblioteca.libro que se pasarán a la plantilla, en nuestro caso todos los registros, o lo que es lo mismo, todos los libros.
Necesitaremos crear un nuevo modelo de tipo AbstractModel.
Editamos el fichero donde tenemos definidos los modelos: models/libro.py, y creamos una nueva clase en Python llamada Fichas_Libros_Report, con el atributo:

_name = 'report.biblioteca.report_imprimir_fichas'

Este atributo es importante porque especifica la clase abstracta que será invocada cuando se ejecute la acción biblioteca.report_imprimir_fichas de categoría ir.actions.report.
Esto lo definimos previamente cuando creamos la acción mediante su record en el fichero reports/informes.xml.

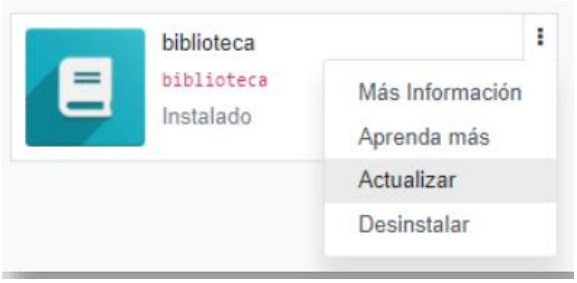
A continuación, creamos el código del método *_get_report_values().*
Como queremos pasar a la plantilla todos los registros del modelo biblioteca.libro, mediante el método search() de *self.env['biblioteca.libro']* construimos una expresión de dominio que seleccione todos los registros. Se recomienda repasar los contenidos expuestos en el tema 10.
Y de igual manera, seleccionamos todos los id de todos los registros del modelo biblioteca.libro.
Y devolvemos los registros, los id de los registros, el modelo al que pertenecen y el data que hemos recibido a través de la invocación del método.
Todo ello será procesado y pasado a la plantilla *biblioteca.report_imprimir_fichas* para generar el informe.

f) La clase Fichas_Libros_Report queda así:

```
class Fichas_Libros_Report(models.AbstractModel):
    _name = 'report.biblioteca.report_imprimir_fichas'
    _description = 'modelo abstracto para imprimir todas las fichas de cada libro'

    def _get_report_values(self, docids=None, data=None):
        docids = self.env['biblioteca.libro'].search([]).ids
        docs = self.env['biblioteca.libro'].search([])
        return {
            'doc_ids': docids,
            'doc_model': self.env['biblioteca.libro'],
            'docs': docs,
            'data': data,
        }
```

Como hemos hecho cambios en los modelos del modulo biblioteca, hay que detener Odoo y volverlo a iniciar.
A continuación, como el módulo ya estaba instalado, hay que actualizarlo en Odoo para que se incorporen los cambios que hemos hecho.
En la página Aplicaciones, Actualizamos la lista de aplicaciones, buscamos el módulo, y en el menú del módulo escogemos actualizar



Como hemos hecho cambios en los modelos del modulo biblioteca, hay que detener Odoo y volverlo a iniciar.
A continuación, como el modulo ya estaba instalado, hay que actualizarlo en Odoo para que se incorporen los cambios que hemos hecho.
En la página Aplicaciones, Actualizamos la lista de aplicaciones, buscamos el módulo, y en el menú del módulo escogemos actualizar:

Nos vamos a la vista formulario de cualquier libro y podremos observar en el botón Imprimir que desplegará un menú con la nueva opción que hemos configurado:

Imprimir | Action

- Imprimir la ficha del libro
- Previsualizar la ficha del libro
- Imprimir todas las fichas

Cambiar descatalogado

El Señor de los Anillos

Descatalogado	
Fecha de publicación	
Páginas	568
Idioma	Español
Resumen	
Autores	

Nombre	
Agregar línea	

Bibliografía y recursos online

https://ubunlog.com/wkhtmltopdf-genera-pdf-imagenes-desde-web/
http://www.geninit.cn/developer/reference/reports.html
https://www.odoo.com/documentation/15.0/developer/reference/backend/actions.html
https://www.odoo.com/documentation/15.0/developer/reference/backend/reports.html
https://www.kite.com/python/docs/ansible.cli.to_text
https://www.odoo.com/documentation/saas-13/reference/orm.html