GENERALITAT VALENCIANA
Conselleria d'Educació, Cultura i Esport

ceedcv CENTRE ESPECÍFIC D'EDUCACIÓ A DISTÀNCIA DE LA COMUNITAT VALENCIANA

UD06. DESARROLLO DE MÓDULOS. INFORMES

Sistemas de Gestión Empresarial
2 Curso // CFGS DAM // Informática y Comunicaciones
Profesor: Alfredo Oltra

Cicles Formatius

ÍNDEX

Versión: 240205.2300

Licencia

Nomenclatura

A lo largo de este tema se utilizarán distintos símbolos para distinguir elementos importantes dentro del contenido. Estos símbolos son:

☑ **Atención**. Importante prestar atención a esta información.

Interesante. Ofrece información sobre algún detalle a tener en cuenta.

1 INTRODUCCIÓN

Una de las necesidades de un ERP es la generaciíonde infromes que muestren de una manera ordenada la información que se puede generar. En *Odoo* esos informes pueden ser generados en HTML o PDF. A efectos del diseñador del informe, el proceso de creación es el mismo, si bien cambian ciertos aspectos técnicos.

De hecho el proceso para la creación del PDF consiste en en la creación del HTML y posteriormente convertirlo a HTML a través de *wkhtmltopdf*, un programa externo (se ejecuta por línea de comandos) que realiza la transformación de manera automática. Es por lo tanto necesaria su instalación en el sistema.

En general el proceso de creación de informes se realiza desde el cliente web, desde un botón (*imprimir*) habilitado en la parte superior de la vista. Este botón lanza una acción que en este caso es de tipo *ir.actions.report*. Este tipo de acciones requiere de un modelo sobre el que trabajar y una plantilla. Esta plantilla, que se asemeja a un documento HTML, es en realidad un documento escrito en *Qweb*, será utilizada para crear el documento HTML que posteriormente será transformado en PDF.

2 ACCIONES

☑ Aunque su lugar de almacenamiento puede ser cualquiera, es conveniente seguir las guías de de diseño de *Odoo* y ubicar tanto la plantillas como las acciones dentro de una carpeta llamada *reports*.

Al igual que las acciones que hemos estudiado, la acción se almacena en la base de datos, por lo tanto debemos crear un fichero XML (que posteriormente cargaremos en el apartado *data* del fichero *__manifest__.py*), en el que definamos el record correspondiente, en este caso para el modelo *ir.actions.report*.

```
<odoo>
  <data>
    <record id="student_pdf_report" model="ir.actions.report">
      <field name="name">Ficha alumno</field>
      <field name="model">school.student</field>
      <field name="report_type">qweb-pdf</field>
      <field name="report_name">school.report_student</field>
      <field name="binding_model_id" ref=”school.student"/>
      <field name="binding_type">report</field>
      <field name="binding_view_types">list,form</field>
      <field name="paperformat_id" ref="modulo_tc.custompaperformat"/>
      <field name="print_report_name">Ficha estudiante - %s' % (object.nia)</field>
    </record>
  </data>
</odoo>
```

Donde:

- *name*: la etiqueta que aparecerá en el menú.
- *report_name*: el *external id* de la plantilla *qweb* que define el formato del informe.

- *model*: el modelo sobre el que vamos a generar el informe
- *report_type*: que indica el tipo de informe a generar. Las opciones son: *qweb-pdf (*formato PDF), *qweb-html* (formato HTML), *qweb-xlsx* (formato XLSX), *qweb-odt* (formato ODT), *qweb-txt* formato TXT).
- *binding_model_id*: la referencia al *external id* del modelo enlazado en el que debe aparecer la opción de impresión del informe (normalmente coincide con el valor de *model* aunque en formatos diferentes).
- *binding_type* (opcional): nombre del menú en el que aparecerá la entrada de la generación del informe. Las opciones son *report* (aparecerá en el menú *Print*) o *action* (menú *Acción*).
- *binding_view_types* (opcional): las vistas del modelo enlazado en el que aparecerá la opción de impresión del informe
- *print_report_name*: nombre del fichero del informe generado.
- *paperformat_id*: la referencia al *external id* fdel formato de papel con el que se ha generar el informe.

Para este caso en particular, es posible utilizar el alias *report* que simplifica la estructura del registro .

```
    <report id="student_pdf_report"
            string="Ficha alumno"
            model="school.student"
            report_type="qweb-pdf"
            name="school.report_student"
            file="school.report_student"
            print_report_name="'Ficha estudiante - %s' % (object.nia)"/>
```

Básicamente permite los mismos atributos cambiando *name* por *string* (el *name* en *<report>* es el nombre de la acción)*, report_name* por *file* y sin la necesidad de los *binding.*

3 TEMPLATE

La creación de la plantilla se puede realizar de manera independiente o incluyéndola junto con la *action*. Dicho de otra manera, se puede optar por crear ficheros *xml* por informes (que incluyen en su interior la plantilla y *action*) o ficheros de *actions* y de *plantillas*. A lo largo de este documento trabajaremos suponiendo que tanto la action como la plantilla están en el mismo fichero.

Existe tambien un alias llamado <*template*> que permite simplificar la estructura del registro.

El alma de la plantilla es *QWeb*, un *template engine* (motor de plantillas). Un *motor engine* es una herramienta muy utilizada en la creación de sitios dinámicos, que lo que permite es que se pueda separar los datos de la presentación de esos datos. Explicado de una manera muy sencilla, lo que permite es la creación de documentos XML que contienen variables e instrucciones (como condicionales o bucles) que una vez son procesados por el motor generan el documento HTML completo preparado para ser renderizado.

Para la definición de la plantilla *QWeb* utiliza el tag <*t*> que, junto con atributos especiales (por ejemplo *t-call* o *t-foreach*) van a permitir definir la estructura del documento.

Una plantilla básica podría ser algo similar a esto:

```
<template id="report_student”>
   <t t-call="web.basic_layout">
     <t t-foreach="docs" t-as=”o”>
        <t t-call="web.external_layout">
          <div class="page">
          </div>
        </t>
     </t>
   </t>
</template>
```

Vamos a analizar cada elemento:

- el *id* es el identificador de la plantilla, *id* que es utilizado en la acción.
- *t-call* permite la llamada a subplantillas, en este caso a la plantilla básica.
- *t-foreach* va a repetir todo lo que tiene en su interior (en este caso simplemente un *div* con la clase *page*) tantas veces como elementos haya en *docs*, llamando a cada elemento *o*. Para entenderlo mejor, si pensaramos en python sería algo similar a `for o in docs.`

3.1 Subplantillas *Odoo*.

Aunque es posible crear plantillas personalizadas, lo habitual es utilizar alguna de las proporcionadas por *Odoo.* Existen muchas disponibles (puedes consultarlas desde el fichero */addons/web/views/report_templates.xml*).

Aunque el número de plantillas es muy elevado, muchas de ellas son de uso interno o subplantillas de otras plantillas y su uso debe realizarse con cuidado ya que pueden generar errores o resultados inesperados.

Entras las habitualmente utilizadas:

- *web.html_container.* Es la plantilla base para documentos *html,* ya que incorpora la estructura mínima aconsejable de un documento de este tipo *(<head>* con *viewport, title... y* el *body* con un elemento de tipo *<main>*)
- *web.basic_layout.* Se basa en la plantilla *html_container* incluyendo un *div* con la clase *article*.
- *web.external_layout*. Básicamente llama a la plantilla *web.external_layout_standar*, que añade una cabecera con el logo y la dirección de la empresa y un *footer* con datos de contacto y el número de página.

Esta última suele ser la opción más habitual, por lo que la plantilla básica resultante queda:

```
<template id="report_student”>
   <t t-call="web.html_container">
     <t t-foreach="docs" t-as=”o”>
        <t t-call="web.external_layout">
          <div class="page">
          </div>
        </t>
     </t>
   </t>
</template>
```

Es importante pensar en el informe como un documento en el que podrían aparecer reflejados muchos registros, por ejemplo las fichas de varios estudiantes. Todos los registros fiorman parte del mismo docuimetno, con lo cuals sólo hay una estructura html (la proporcionada por *web.html_container*) pero cada una de esas fichas tendría una cabecera y un footer (que es el que aporta *web.external_layout).* Es por ello que *web.html_container* está fuera del *foreach* y sin embargo *web.external_layout* se llama dentro, es decir, para cada registro.

3.2 Definiendo la estructura

Una vez cargadas las subplantillas (y por lo tanto definida la estructura más general) hay que definir la estructura que queremos darle a cada registro utilizando etiquetas *html* junto con el apoyo de etiquetas *QWeb*. Además es posible utilizar clases css, incluso *Bootstrap* para los estilos. De hecho, lo más habitual es replantear el layout de la página utilizando el grid de este framework.

wkhtmltopdf no tiene soporte completo de layouts nativos de css como *flex* o *css-grid*. Es por eso que en el caso de que el objetivo sea generar el informe en formato pdf se busque otras opciones externas como *Bootstrap grid* o más sencillas como posicionamientos *position* o *float.*

3.3 Etiquetas *QWeb*

QWeb dispone de muchas más opciones que pueden consultarse desde el sitio web de documentación de Odoo.

t-esc

El atributo *t-esc* evalúa una expresión y vuelca el resultado en el documento.

```
<p><t t-esc="o.surname"/></p>
```

que renderizará el valor de *surname* del registro *o.* Por ejemplo:

```
<p>Pérez García</p>
```

t-out escapa el contenido para evitar problemas de seguridad. En caso de querer mostrar contenido html crudo (es decir que la variable a mostrar sea código HTML a ser renderizado) se recomienda utilizar la librería *markupsafe.Markup*

Al igual que las acciones que hemos estudiado, la acción se almacena en la base de datos, por

t-if, t-elif, t-else

Los atributos *t-if, t-elif, t-else* permiten que se rendericen partes en función de condiciones

```
<div>
   <p t-if="o.is_head== True">Jefe de estudios</p>
   <p t-elif="o.is_coord== True">Coordinador</p>
   <p t-else="">Profesor</p>
<div>
```

t-foreach

Realiza iteraciones a través de todos los elementos definidos en la variable

```
<t t-foreach="o.subjects" t-as=”subject”>
   <p t-out="subject.abbr"></p>
</t>
```

que renderizará algo similar a:

```
<p>SGE</p>
<p>PSP</p>
<p>DI</p>
```

t-foreach proporciona además una serie de variables para poder usar dentro del bucle, por ejemplo `$as_index` nos proporciona el índice de la iteración.

```
<t t-foreach="o.subjects" t-as=”subject”>
   <p t-out="subject.abbr"> (<span t-out="$as_index"></span>)</p>
</t>
```

renderizará

```
<p>SGE (0)</p>
<p>PSP (1)</p>
<p>DI (2)</p>
```

t-attf-$name

t-attf-$name permite la creación de un atributo (de nombre $name) en función de una expresión.

```
<p t-attf-class="row {{o.is_head == True ? ‘red’ : ‘blue’ }}”></p>
```

Al renderiozarse, suponiendo que *is_head* es True, generará algo como:

```
<p class="row red”></p>
```

t-set

t-set permite la asignación de variables para su uso interno. La asignación se pude realizar de dos maneras, con y sin *t-value*. Por ejemplo, para crear una variable llamada variable y asignarle el valor 34:

```
<t t-set="variable” t-value="34”/>
```

O, para almacenar en la variable un párrafo *html*:

```
<t t-set="variable”>
   <p>Lista de Tareas</p>
</t>
```

La variable mágica 0

La variable *0* representa a todos los elementos incluidos dentro de una llamada a una plantilla. De esa manera es posible adaptar la plantilla a las necesidades puntuales. Por ejemplo, si tenemos una plantilla llamada *mi_plantilla*:

```
<div>
   <p>Lista de Tareas</p>
   <t t-out="0"/>
</div>
```

Y la llamamos:

```
<t t-call="mi_plantilla">
   <p>Tareas de SGE</p>
</div>
```

Generaría algo como:

```
<div>
  <p>Lista de Tareas</p>
  <p>Tareas de SGE</p>
</div>
```

3.4 Estilos

A la hora de incluir estilos existen dos opciones.

- Embeberlos directamente en el código utilizando la etiqueta `style`:

```
<div class="row" style="border-top: 1px solid black;">
```

- Añadiendo la ubicación de un/os ficheros *css* externos

```
<xpath expr="//style" position="after">
  <link rel="stylesheet" ref="/maya_core/static/css/school_calendar_report.css"/>
</xpath>
```

4 MODIFICACIÓN DE PLANTILLAS *ODOO*

Es posible que queramos modificar parte de la estructura que nos proporciona algunas de la plantillas predefinidas, especialmente la *external_layout/external_layout_standard*. Por ejemplo, es habitual querer modificar la cabecera o el pie de página.

La solución pasa por heredar la plantilla original y adaptarla a nuestras necesidades.

☑ Evidentemente, para ello lo primero que necesitamos es tener el código de la plantilla original y poder estudiarlo. En el fichero *addons/web/views/report_templates.xml* de la instalación de *Odoo* podemos encontrar esas plantillas, aunque también es posible localizarlas en el código disponible en *github*. Por ejemplo, para la versión 17, el código de la plantilla *external_layout_standard* se puede ver aquí.

Por ejemplo, en el caso de que queramos simplificar el *footer* original*:*

```
<odoo>
  <template id="external_layout_standard_report_footer_custom"
inherit_id="web.external_layout_standard">
   <xpath expr="(//div/div[hasclass('row')])[3]" position="replace">
      <div class="col-12" name="company_address">
        <span t-field="company.name" />
        <span t-field="company.street" />
        <span t-field="company.city" />
        <span t-field="company.zip" />
      </div>
    </xpath>
  </template>
</odoo>
```

5 PARSERS

Los *parsers* son modelos (en este caso abstractos) que permiten preprocesar los datos de un modelo antes de ser enviados al informe. Su uso es muy variado, desde una conversión o una validación hasta algo más complejo como el cambio de los datos a otros formatos. Por ejemplo, a partir de los datos de uno de los registros del modelo mostrar un gráfico de cierta complejidad.

Para que puedan ser localizados por *Odoo* a la hora de generar el informe estos modelos se deben llamar (variable `_name`) *report.[atributo name del template]*. Por ejemplo en el caso anterior, que el *id* del *template* era *report_student* el nombre del modelo debería ser: *report.school.report_student*.

Por otra parte el modelo debe tener al menos la función `_get_report_values(self, docids, data)` que es el función que espera encontrar *Odoo* a la hora de generar el informe. Es en esta función donde implementaremos toda la lógica que necesitemos para realizar esa conversión de datos.

Su objetivo es devolver un objeto diccionario con, como mínimo, 3 campos obligatorios: *doc_ids* (los identificadores de los registros a imprimir en el informe), *doc_model* (el modelo sobre el que se trabaja) y *docs* (cada uno del los registros), más todos aquellos que se puedan necesitar en el *template* del report.

```
from odoo import models

class StudentReport(models.AbstractModel):
  _name = 'report.school.report_student'
  _description = 'Parser report calendario escolar'

  def _get_report_values(self, docids, data=None):
    _logger.info("Parser generación calendario escolar")

    docs = self.env['school.student'].browse(docids)

    for doc in docs:
      data_extra[doc.id] = []

      # hacemos algo con los datos
      # en este caso la suma del nombre y del nia separados por ---
      # es un ejemplo no real
      # (si fuera necesario esto sería más sencillo crear un compute field)
      data = doc.name  + "----" + doc.nia

      data_extra[doc.id].append(data)

    # se devuelve lo que interese que aparezca en el template
    return {
      'doc_ids': docids,                                    # obligatorio
      'doc_model': 'school.student',                        # obligatorio
      'docs': docs,                                         # obligatorio
      'data_extra': data_extra
    }
```

6 BIBLIOGRAFÍA

1. Documentación de Odoo