

UD06. DESARROLLO DE MÓDULOS. VISTA

Sistemas de Gestión Empresarial

2 Curso // CFGS DAM // Informática y Comunicaciones

Profesor: Alfredo Oltra

**Cicles
Formatius**

ÍNDIX

1 INTRODUCCIÓN	4
2 LAYOUT <i>TREE</i>	7
2.1 Colores en las líneas	8
2.2 Líneas editables	8
2.3 Campos invisibles	8
2.4 Cálculo de agregados	9
3 LAYOUT <i>FORM</i>	10
3.1 Como definir una vista <i>tree</i> específica en los <i>X2many</i>	13
3.2 Widgets	13
3.3 Valores por defecto en los <i>One2many</i>	15
3.4 Domains en los <i>Many2one</i>	15
3.5 Formularios dinámicos	15
4 LAYOUT <i>CALENDAR</i>	17
5 LAYOUT <i>GRAPH</i>	18
6 LAYOUT <i>SEARCH</i>	19
7 ACCIONES	20
7.1 Windows action	20
8 BIBLIOGRAFIA	22
9 AUTORES	22

Versión: 240107.0102


Licencia




Reconocimiento – NoComercial – CompartirIgual (by-nc-sa). No se permite un uso comercial de la obra original ni de las posibles obras derivadas, la distribución de las cuales se debe hacer con una licencia igual a la que regula la obra original.

Nomenclatura

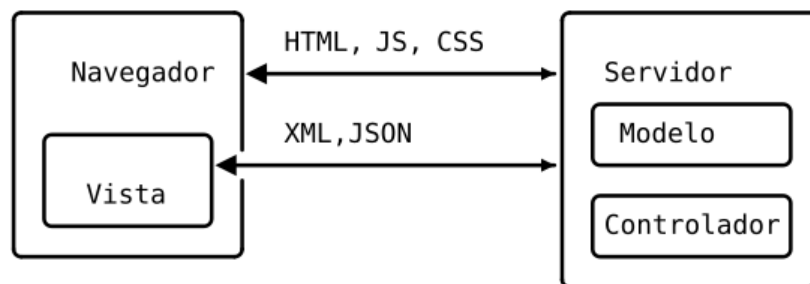
A lo largo de este tema se utilizarán distintos símbolos para distinguir elementos importantes dentro del contenido. Estos símbolos son:

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

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

1 INTRODUCCIÓN

En el esquema *Modelo-Vista-Controlador* que sigue *Odoo*, la vista se encarga de todo lo que tiene que ver con la interacción con el usuario. En *Odoo*, la vista es un programa completo de cliente en Javascript que se comunica con el servidor con mensajes breves. La vista tiene tres partes muy diferentes: el *backend*, el *frontend* (la web) y el *TPV (POS)*. Nosotros vamos a centrarnos en la vista del *backend*.



En la primera conexión con el navegador web, el servidor *Odoo* le proporciona un HTML mínimo, una SPA (*Single Page Application*) en Javascript y un CSS. Esto es un cliente web para el servidor y es lo que se considera la vista.

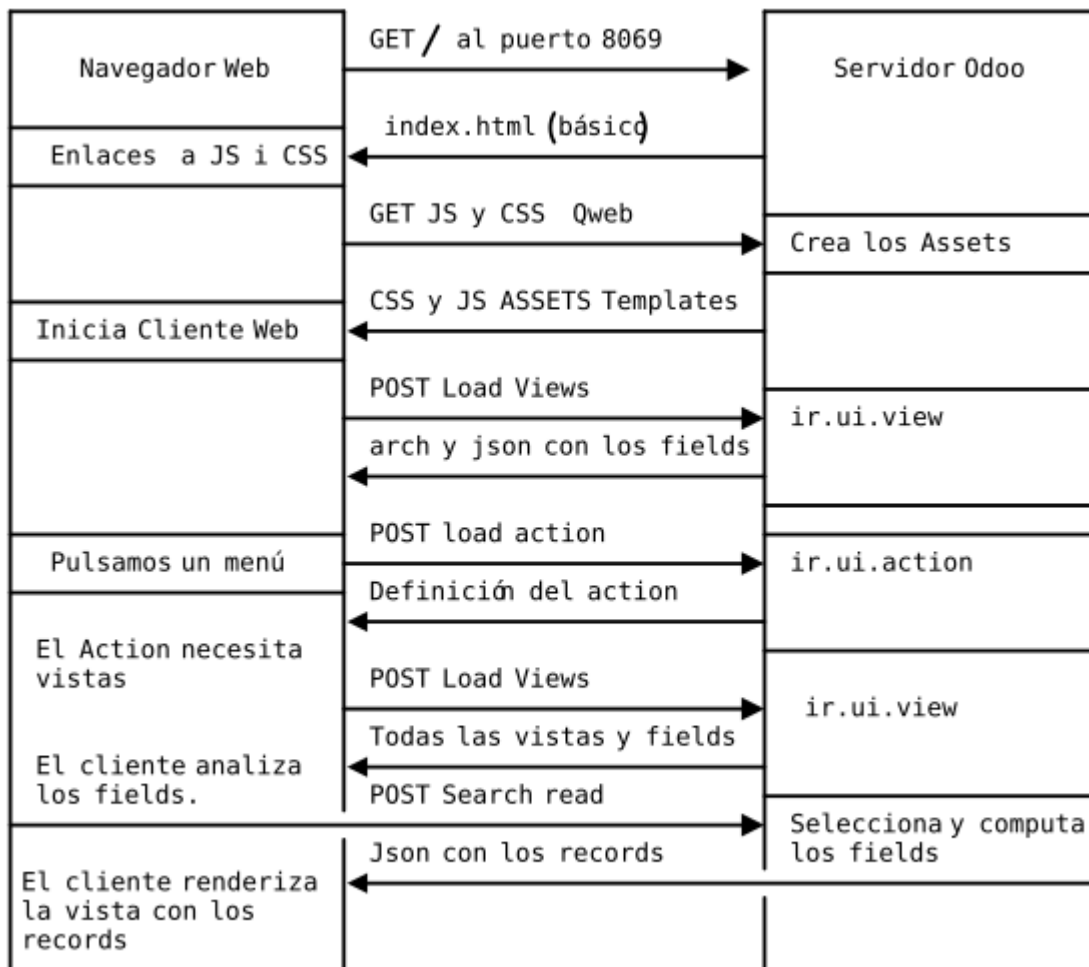
Pero *Odoo* tampoco carga la vista completa, ya que podría ser inmensa. Cada vez que los menús o botones de la vista requieren cargar la visualización de unos datos, piden al servidor un XML que defina cómo se van a ver esos datos y un JSON con los datos. Entonces la vista renderiza los datos según el esquema del XML y los estilos definidos en el cliente.

Esta visualización se hace con unos elementos llamados *widgets*, que son combinaciones de CSS, HTML y Javascript que definen el aspecto y comportamiento de un tipo de datos en una vista en concreto. Todo esto es lo que vamos a ver con detalle en este apartado.

Los XML que definen los elementos de la vista se guardan en la base de datos y son consultados como cualquier otro modelo. De esta manera se simplifica la comunicación entre el cliente web y el servidor y el trabajo del controlador. Puesto que cuando creamos un módulo, queremos definir sus vistas, y esas vistas se definen en XML, debemos crear archivos de este tipo. Estos serán referenciados en el `__manifest__.py` en el apartado *data* y serán cargados e introducidos en la correspondiente tabla de la base de datos cuando se instale o actualice el módulo.



Cuando creamos un módulo con *scaffold*, en el `__manifest__.py` se crea una referencia a un *xml* del directorio *views*. Este *xml* contiene un ejemplo (completamente conectado) de los principales elementos de las vistas, que veremos a continuación.



La vista tiene varios elementos necesarios para funcionar:

- **Vistas.** Son las propias definiciones de las vistas, guardadas en el modelo `ir.ui.view`. Estos elementos tienen al menos los *fields* que se van a mostrar y pueden tener información sobre la disposición, el comportamiento o el aspecto de esos *fields*.
- **Menus.** Están distribuidos de forma jerárquica y se guardan en el modelo `ir.ui.menu`.
- **Actions.** Las acciones o *actions* enlazan una acción del usuario (como pulsar en un menú) con una llamada al servidor desde el cliente para pedir algo (como cargar una nueva vista). Las *actions* están guardadas en varios modelos dependiendo del tipo.

Un ejemplo completo podría ser:

```
<odoo>
<data>
  <!-- explicit list view definition -->
  <record model="ir.ui.view" id="prueba.student_list">
    <field name="name">Student list</field>
    <field name="model">prueba.student</field>
```

```
<field name="arch" type="xml">
  <tree>
    <field name="name"/>
    <field name="topics"/>
  </tree>
</field>
</record>

<!-- actions opening views on models -->
<record model="ir.actions.act_window" id="prueba.student_action_window">
  <field name="name">student window</field>
  <field name="res_model">prueba.student</field>
  <field name="view_mode">tree,form</field>
</record>

<!-- Top menu item -->
<menuitem name="prueba" id="prueba.menu_root"/>
<!-- menu categories -->
<menuitem name="Administration" id="prueba.menu_1" parent="prueba.menu_root"/>
<!-- actions -->
<menuitem name="Students" id="prueba.menu_1_student_list"
  parent="prueba.menu_1" action="prueba.student_action_window"/>
</data>
</odoo>
```

En este ejemplo se ven *records* en XML que indican que se va a guardar en la base de datos. Estos *records* dicen el modelo donde se guardará y la lista de *fields* que queremos guardar.

El primer *record* define una vista (layout) tipo *tree* que es una lista de estudiantes donde se verán los campos *name* y *topics*. El segundo *record* es la definición de un *action* tipo *window*, es decir, que abre una ventana para mostrar unas vistas de tipo *tree* y *form* (formulario). Los otros definen tres niveles de menú: el superior, el intermedio y el menú desplegable que contiene el *action*. Cuando el usuario navegue por los dos menús superiores y presione el tercer elemento de menú se ejecutará ese *action* que cargará la vista *tree* definida y una vista *form* inventada por *Odoo*.



Para poder observar un modelo en el cliente web de *Odoo* no necesitamos más que un menú que accione un *action* tipo *window* sobre ese modelo. *Odoo* es capaz de inventar las vistas básicas que nos permiten observarlo. No obstante suelen ser menos atractivas y útiles que las que definimos nosotros.

Quedémonos por el momento con la definición básica de un *action window* y de los menús. Vamos a centrarnos antes en las vistas.

2 LAYOUT TREE

La vista tree muestra una lista de *records* sobre un modelo. Veamos un ejemplo básico:

```
<record model="ir.ui.view" id="prueba.ciclos_list">
  <field name="name">Lista de Ciclos</field>
  <field name="model">prueba.ciclo</field>
  <field name="arch" type="xml">
    <tree>
      <field name="name"/>
      <field name="abbr"/>
      <field name="code"/>
      <field name="abbr"/>
      <field name="subjects_ids"/>
    </tree>
  </field>
</record>
```

Ciclos			
<div> <div>Crear </div> <div> <div>Buscar...</div> <div>Q</div> </div> <div> <div>▼ Filtros</div> <div>≡ Agrupar por</div> <div>★ Favoritos</div> </div> <div>1-11 / 11 < ></div> </div>			
<input type="checkbox"/> Ciclo	Abreviatura	Código	Módulos
<input type="checkbox"/> Administración y Finanzas	AFI	441104	18 registros
<input type="checkbox"/> Asistencia a Dirección	ADI	906104	17 registros
<input type="checkbox"/> Desarrollo de Aplicaciones Web	DAW	845104	17 registros
<input type="checkbox"/> Desarrollo de Aplicaciones Multiplataforma	DAM	836104	16 registros
<input type="checkbox"/> Administración de Sistemas Informáticos y Redes	ASIR	829104	17 registros
<input type="checkbox"/> Gestión de Alojamientos Turísticos	GAT	714104	17 registros
<input type="checkbox"/> Guía, Información y Asistencias Turísticas	GIAT	828104	17 registros
<input type="checkbox"/> Agencias de Viajes y Gestión de Eventos	AVGE	827104	17 registros
<input type="checkbox"/> Comercio Internacional	CI	449104	17 registros
<input type="checkbox"/> Transporte y Logística	TIL	898104	16 registros
<input type="checkbox"/> Gestión y Ventas de Espacios Comerciales	GEVEC	899104	17 registros

Como se puede observar, esta vista se guardará en el modelo *ir.ui.view* con un external ID llamado *prueba.ciclos_list*. El modelo tiene más posibles *fields*, pero los mínimos necesarios son *name*, *model* y *arch*. El *field arch* guarda el XML que será enviado al cliente para que renderice la vista. Dentro del *field arch* está la etiqueta *<tree>* que indica que la disposición de los elementos será en forma de lista y dentro de esta etiqueta tenemos más *fields* (en este caso los *fields* del modelo *prueba.ciclo*) que queremos mostrar.

Esta vista *tree* se puede mejorar, con muy poco esfuerzo, de muchas formas. Veamos algunas de ellas.

2.1 Colores en las líneas

Odoo no da, de manera sencilla, libertad absoluta al desarrollador en este aspecto y permite un número limitado de estilos para dar a las líneas en función de alguna condición. Estos estilos son similares a los de *Bootstrap*:

- *decoration-bf*: líneas en BOLD
- *decoration-it*: líneas en ITALICS
- *decoration-danger*: color LIGHT RED
- *decoration-info*: color LIGHT BLUE
- *decoration-muted*: color LIGHT GRAY
- *decoration-primary*: color LIGHT PURPLE
- *decoration-success*: color LIGHT GREEN
- *decoration-warning*: color LIGHT BROWN

Por ejemplo, podemos decorar la lista en función de un campo llamado *state* :

```
<tree decoration-info="state=='draft'" decoration-danger="state=='trashed'">
```



Se puede comparar un *field date* con una variable de *QWeb* llamada *current_date*:

```
<tree decoration-info = "start_date == current_date">
```

O, incluso, podemos decorar uno de los campos de la lista en función *state* :

```
<field name="state" decoration-danger="state == 'trashed'">
```

2.2 Líneas editables

Por defecto, si el usuario tiene permisos de edición sobre el modelo, al pulsar sobre un elemento de una lista, se abrirá un formulario con los datos de esa entrada. Sin embargo, en algunas ocasiones únicamente queremos editar algún campo de una manera más rápida y sencilla. Para ello, es posible hacer el *tree* editable.

Para hacerlo editable hay que añadir al *tree* el atributo *editable='[top | bottom]'*. La diferencia entre *top* y *bottom* es el lugar en la lista (arriba o abajo) donde se mostrará la línea de inserción de nuevos registros.

2.3 Campos invisibles

Algunos campos solo han de estar para definir el color de la línea, servir como lanzador de un *field computed* o ser buscados, pero el usuario no necesita verlos. Para ello hay varias opciones:

- poner el atributo *invisible="1"* en el *field* que necesitemos.

```
<field name="state" invisible="1">
```


- Indicándolo en el atributo *attrs*

```
<field name="state" attrs="{ 'invisible': 1 }">
```

En estos casos la columna que define el campo sigue apareciendo (sólo se oculta el valor del campo), por lo que si lo que queremos es ocultarla, debemos utilizar el atributo *column_invisible*:

```
<field name="state" attrs="{ 'column_invisible': [True] }">
```

En ambos casos es posible aplicar condiciones para resolver si el campo debe ser invisible o no a través de un dominio de búsqueda. Por ejemplo:

```
<field name="state" attrs="{ 'invisible': [ '|', ('state', '=', '2'), ('state', '=', '1') ] }">
```

2.4 Cálculo de agregados

En los *fields* numéricos, es posible mostrar en una fila extra, valores agregados como la suma o la media. Para ello debemos usar el atributo *sum* o *avg* en el campo correspondiente. Un ejemplo de como quedaría una vista *tree* que mostrara por cada estudiante su nota (poniendo la línea en azul en caso de estar suspendido) y mostrando al final una celda con la media de todas las notas de todos los estudiantes sería algo similar a:

```
<record model="ir.ui.view" id="prueba.estudiante_list">
  <field name="name">Lista Estudiantes</field>
  <field name="model">prueba.estudiante</field>
  <field name="arch" type="xml">
    <tree decoration-info="qualification<5" editable="top">
      <field name="name"/>
      <field name="topics" invisible="true"/>
      <field name="qualification" avg="Media"/>
    </tree>
  </field>
</record>
```

3 LAYOUT FORM

Esta vista permite editar o crear un nuevo registro en el modelo que represente. Muestra un formulario que tiene versión editable y versión no editable. Al tener dos versiones y necesitar más complejidad, la vista *form* tiene muchas más opciones.



En esta vista hay que tener en cuenta que al final se traducirá en elementos HTML y CSS y que los selectores CSS son estrictos con el orden y jerarquía de las etiquetas. Por tanto, no todas las combinaciones funcionan siempre

Odoo por defecto crea formularios adaptados en función de la lógica interna que proporciona la aplicación SPA, pero también permite cierta libertad al desarrollador para controlar la disposición de los *fields* y la estética. No obstante, hay un esquema que conviene seguir.

Un formulario puede ser definido con la etiqueta `<form>` con etiquetas de *fields* dentro, igual que el *tree*. Pero conseguir un buen resultado será más complicado y hay que introducir elementos HTML. Para simplificar, Odoo propone unos contenedores con unos estilos predefinidos que funcionan bien y estandarizan los formularios de toda la aplicación.

Sheet

Utilizando la etiqueta *sheet* dentro de *form* los campos se apilan. Si la utilizamos, los *fields* perderán la etiqueta que los identifica (la asociada al parámetro *string*). Para poder verlo hay dos opciones:

- utilizar la etiqueta `<label for="field">`. Esta etiqueta únicamente incluye el texto definido en el parámetro *string* del campo *field* en el documento HTML, en la misma posición que se indica.
- utilizar la etiqueta `<label>Texto</>`. A diferencia de la anterior, permite añadir un *Texto* en el documento HTML, en la misma posición que se indica.
- agrupar los *fields* dentro de etiqueta `<group string="Nombre del grupo">`. De esta manera todos los *fields* incluidos mostraran de nuevo el valor definido en el parámetro *string*. Además podrá incluirse un cabecera como nombre del grupo.

Si hacemos varios *groups* o *groups* dentro de *groups*, el CSS de Odoo ya alinea los *fields* en columnas o los separa correctamente. Sin embargo si queremos separar manualmente algunos *fields*, podemos utilizar la etiqueta `<separator string="Texto a mostrar">`.

```
<record model="ir.ui.view" id="prueba.local_form">
  <field name="name">Información del local</field>
  <field name="model">prueba.local</field>
  <field name="arch" type="xml">
    <form>
      <sheet>
        <group string="Información básica">
          <group>
            <field name="code"/>
          </group>
        </group>
      </sheet>
    </form>
  </field>
</record>
```

```

        <field name="city"/>
    </group>
    <group>
        <field name="address"/>
        <separator />
        <field name="area"/>
    </group>
</group>
<label for="capacity"/>
<field name="capacity"/>
</sheet>
</form>
</field>
</record>

```

Información básica

Código

Dirección

Ciudad

Valencia

Superficie

0,00

Capacity

0



El elemento separator puede incluir dos atributos: string, que permite incluir una cadena de texto y style, que permite que se le aplique un estilo css determinado. Este último puede ser muy útil para, por ejemplo, modificar la separación a más o menos espacio: `<separator string="Otros datos" style="padding-bottom:15px"/>`

Notebook

Otro elemento de separación y organización es `<notebook>` `<page string="título">`, que crea unas pestañas que esconden partes del formulario y permiten que quepa en la pantalla.

```

<record model="ir.ui.view" id="prueba.local_form">
    <field name="name">Información del local</field>
    <field name="model">prueba.local</field>
    <field name="arch" type="xml">
        <form>
            <group>
                <group>

```

```
<field name="code"/>
</group>
<group>
  <field name="city"/>
</group>
</group>
<notebook>
  <page string="Página 1">
    <label for="address"/>
    <field name="address"/>
  </page>
  <page string="Página 2">
    <label for="capacity"/>
    <field name="capacity"/>
  </page>
</notebook>
</form>
</field>
</record>
```

Código		Ciudad	Valencia
Página 1 Página 2			
Dirección			

Header

La etiqueta *header* permite la creación de una cabecera en el formulario ya sea para mostrar información o para añadir algún tipo de control.

```
<record model="ir.ui.view" id="prueba.local_form">
  <field name="name">Información del local</field>
  <field name="model">prueba.local</field>
  <field name="arch" type="xml">
    <form>
      <header>
        <button name="my_button" icon="fa-cog" string="Mandar mensajes socios"
          class="oe_highlight" type="object"/>
      </header>
      <group>
        ...
      </group>
    </form>
  </field>
</record>
```

Guardar Descartar

Mandar mensaje socios

Código Ciudad

Página 1 Página 2

Dirección



Las combinaciones de *group*, *label*, *separator*, *notebook* y *page* son muchas. Se recomienda ver cómo han hecho los formularios en algunas partes de *Odoo*. Los formularios oficiales tienen muchas cosas más complejas.

3.1 Como definir una vista *tree* específica en los *X2many*

Los *One2many* y *Many2many* se muestran, por defecto, dentro de un formulario como una subvista *tree*. *Odoo* coge la vista *tree* con más prioridad del modelo al que hace referencia el *X2many* y la incrusta dentro del formulario. Esto provoca dos problemas:

- Si esa vista de referencia cambia, también cambian los formularios.
- Las vistas *tree*, cuando son independientes, suelen mostrar más *fields* de los que se necesitan dentro del un formulario que las referencia.

Por eso se puede definir un *tree* dentro del *field*:

```
<field name="subscriptions" colspan="4">
  <tree>...</tree>
</field>
```

3.2 Widgets

Un *widget* es un componente del cliente web que sirve para representar un dato de una forma determinada. Un *widget* tiene una plantilla HTML, un estilo con CSS y un comportamiento definido con Javascript. Si queremos, por ejemplo, mostrar y editar fechas, *Odoo* tiene un *widget* para los *Datetime* que muestra la fecha con formato de fecha y muestra un calendario cuando estamos en modo edición.

Además, algunos *fields* pueden mostrarse con distintos *widgets* en función de lo que queramos. Por ejemplo, las imágenes por defecto están en un *widget* que permite descargarlas, pero no verlas en la web. Si le ponemos `widget='image'` las mostrará.

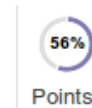


Es posible hacer nuestros propios *widgets*, sin embargo, requiere tener conocimientos sobre como modificar el cliente web, lo cual no está contemplado en esta unidad didáctica.

La lista de *widgets* disponibles para su uso es muy grande y es dependiente de la versión de *Odoo* para la que estamos desarrollando. Dar una explicación detallada de cada uno de ellos está fuera del alcance de la unidad, pero algunos de los más genéricos pueden ser:

Fields numéricos

- **integer**: el número sin comas. Si está vacío, muestra un 0.
- **char**: el carácter, aunque muestra el campo más ancho. Si está vacío muestra un hueco.
- **id**: no se puede editar.
- **float**: el número con decimales.
- **percentpie**: un gráfico circular con el porcentaje.
- **monetary**: con dos decimales.
- **field_float_rating**: estrellas en función de un float.
- **progressbar**: una barra de progreso



Fields texto

Para los *fields* de texto tenemos algunos que con su nombre se explican solos: *char*, *text*, *email*, *url*, *date*, *html*.

Fields boolean

Para los booleanos, a partir de *Odoo 13* se puede mostrar una cinta al lado del formulario con el *widget* llamado *web_ribbon*.



Fields relacionales

Los *fields* relacionales se muestran por defecto como un *selection* o un *tree*, pero pueden ser:

- **many2onebutton**: indica solamente si está seleccionado.
- **many2many_tags**: que muestra los datos como etiquetas.

Id player ●

Reforc

grup1 x grup3 x



La lista de *widgets* es muy larga y van entrando y saliendo en las distintas versiones de *Odoo*. Es recomendable explorar los módulos oficiales y ver cómo se utilizan para copiar el código en nuestro módulo. Hay muchos más, algunos únicamente están si has instalado un determinado módulo, porque se hicieron a propósito para ese módulo, aunque los puedes aprovechar si lo pones como dependencia.



Es posible encontrar en internet listas de los diferentes widgets disponibles para cada versión. Por ejemplo, podemos encontrar para la versión 12, 14 o 16.

3.3 Valores por defecto en los One2many

Cuando tenemos un *One2many* en una vista *form*, nos da la opción de crear nuevos registros. Recordemos que un *One2many* no es más que una representación del *Many2one* que hay en el otro modelo. Por tanto, si creamos nuevos registros, necesitamos que el *Many2one* del modelo a crear referencie al registro que estamos modificando del modelo que tiene el formulario.

Para conseguir que el formulario que sale en la ventana emergente tenga ese valor por defecto, lo que haremos será pasar por contexto un dato con un nombre especial que será interpretado:

```
context={'default_<nombre del field many2one>':active_id}.
```

Esto se puede hacer también en un *action*. De hecho, al pulsar un elemento del *tree* se ejecuta un *action* también:

```
<field name="context">{"default_doctor": active_id}</field>
```

La palabra reservada `active_id` es una variable que apunta al *id* del registro que está activo en ese formulario.



El concepto de contexto en *Odoo* no está explicado todavía. De momento pensemos que es un cajón de sastre donde poner las variables que queremos pasar de la vista al controlador y viceversa. .



Dos fuentes interesantes donde poder obtener información sobre el uso de context son [cyborsys](#) y [odootricks](#)

3.4 Domains en los Many2one

Aunque se pueden definir en el modelo, puede que en una vista determinada necesitemos un *domain* más específico.

```
<field name="hotel" domain="[('ishotel', '=', True)]"/>
```

3.5 Formularios dinámicos

El cliente web de *Odoo* es una web reactiva. Esto quiere decir que reacciona a las acciones del usuario o a eventos de forma automática. Parte de esta reactividad se puede definir en la vista *form* haciendo que se modifique en función de varios factores. Esto se consigue con el atributo *attrs* entre otros de los *fields*.

Se puede ocultar condicionalmente un *field*:

```
<field name="boyfriend_name" attrs="{ 'invisible': [('married', '!=', False)]}" />
```

Se puede mostrar u ocultar en modo edición o lectura:

```
<field name="partido" class="oe_edit_only"/>
<field name="equipo" class="oe_read_only"/>
```

Muchos formularios tienen estados y se comportan como un asistente. En función de cada estado se pueden mostrar u ocultar *fields*. Hay un atajo al ejemplo anterior si tenemos un *field* que específicamente se llama *state*. Con el atributo *states* se puede mostrar u ocultar elementos de la venta:

```
<group states="i,c,d">
  <field name="name"/>
</group>
```

También existe la opción de ocultar una columna de un *tree* de un *X2many*:

```
<field name="lot_id" attrs="{ 'column_invisible': [('parent.state', 'not in', ['sale',
'done'])] }"/>
```



En el ejemplo anterior, *parent.state* hace referencia al campo *state* del modelo padre de ese *tree*. Hay que tener en cuenta que ese *tree* se muestra con el modelo al que hace referencia el *X2many*, pero está dentro de un formulario de otro modelo.

Dentro de los formularios dinámicos, se puede editar condicionalmente un *field*. Esto quiere decir que permitirá al usuario modificar un *field* en función de una condición:

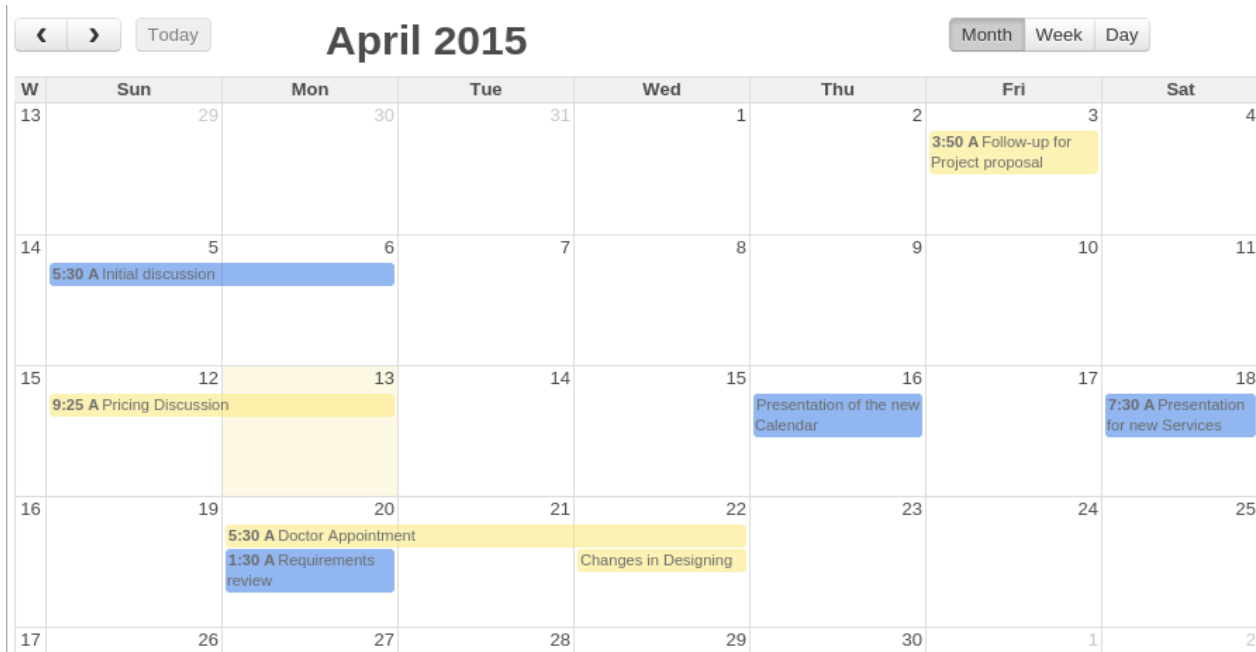
```
<field name="name2" attrs="{ 'readonly': [('condition', '=', False)] }"/>
```

Veamos ahora un ejemplo con todos los *attrs*:

```
<field name="name" attrs="{ 'invisible': [('condition1', '=', False)],
  'required': [('condition2', '=', True)],
  'readonly': [('condition3', '=', True)] }" />
```


4 LAYOUT CALENDAR

Esta vista muestra un calendario si los datos de los registros del modelo tienen al menos un *field* que indica una fecha y otro que indique una fecha final o una duración.



La sintaxis es muy simple:

```
<record model="ir.ui.view" id="school.travel_calendar">
  <field name="name">travel calendar</field>
  <field name="model">school.travel</field>
  <field name="arch" type="xml">
    <calendar string="School Calendar" date_start="init_time"
      date_delay="duration" <!-- Puede ser delay (en horas) o date_stop -->
      color="origin_school" <!-- El color indica el field que lo modifica
        No un color literalmente -->
      <field name="name"/>
    </calendar>
  </field>
</record>
```

Como atributos, entre otros:

- *date_start*: Indica la fecha de inicio del evento de calendario.
- *date_stop*: Indica la fecha de finalización del evento de calendario.
- *date_delay*: Indica el tiempo en horas para un registro. Tiene mayor preferencia que *date_stop*. Si se especifica *date_delay*, se ignorará *date_stop*.
- *mode*: La vista de calendario proporciona una vista diaria, semanal y mensual. Indica el modo de vista predeterminado de la vista Calendario cuando se carga la página. Los valores para este atributo pueden ser *day*, *week* y *month*.+

- **color**: Indica el campo que definirá el color. Todos los elementos con el mismo valor en ese campo se mostrarán del mismo color
- **duración_día**: Un valor entero que ayuda a indicar la duración de la jornada laboral. Por defecto son 8 horas.
- **form_view_id**: Permite sobrescribir la vista de formulario que se abrirá cuando pulsemos sobre el botón editar del popup que aparece al pulsar sobre el evento.
- **event_open_popup**: Si esta opción está activada (*true*), el evento se abrirá como una ventana emergente.
- **quick_add**: permite la creación rápida de eventos, evitando el formulario completo del modelo.

5 LAYOUT GRAPH

La vista *graph* permite mostrar una gráfica a partir de algunos *fields* numéricos que tenga el modelo. Esta vista muestra tres tipos: *bar* (gráfico de barras, que es tipo de gráfico que se mostrará por defecto), *pie* (gráfico de tarta), o *line* y se comporta **agregando los valores** que ha de mostrar.

En este ejemplo se crea un gráfico en el que se muestra, la evolución de las notas con el tiempo de cada estudiante. Se ubica en el eje X el tiempo, en el Y el estudiante y como medida la calificación. El gráfico por defecto a mostrar será el de líneas, aunque se podrá visualizar el gráfico de tarta o el de barras desde la propia vista.

```
<record model="ir.ui.view" id="school.qualifications_graph">
  <field name="name">Qualifications graph</field>
  <field name="model">school qualifications</field>
  <field name="arch" type="xml">
    <graph string="Calificaciones" type="line">
      <field name="student"/>
      <field name="time"/>
      <field name="qualification" type="measure"/>
    </graph>
  </field>
</record>
```

6 LAYOUT SEARCH

Esta no es una vista como las que hemos visto hasta ahora. Entra dentro de la misma categoría y se guarda en el mismo modelo, pero no se muestra ocupando la ventana, sino la parte superior donde se sitúa el formulario de búsqueda. Lo que permite es definir los criterios de búsqueda, filtrado o agrupamiento de los registros que se muestran en el cliente web.

Veamos un ejemplo completo de vista *search*:

```
<search>
  <field name="name"/>
  <field name="inventor_id"/>
  <field name="description" string="Name and description" filter_domain="['|', ('name',
'ilike', self), ('description', 'ilike', self)]"/>
  <field name="boxes" string="Boxes or @" filter_domain=    ['|', ('boxes', '=', self),
('kg', '=', self)]"/>
  <filter name="my_ideas" string="My Ideas" domain="[('inventor_id', '=', uid)]"/>
  <filter name="more_100" string="More than 100 boxes" domain="[('boxes', '>', 100)]"/>
  <filter name="Today" string="Today" domain="[('date', '>=',
datetime.datetime.now().strftime('%Y-%m-%d 00:00:00')), ('date', '<=',
datetime.datetime.now().strftime('%Y-%m-%d 23:23:59'))]"/>
  <filter name="group_by_inventor" string="Inventor" context="{ 'group_by':
'inventor_id' }"/>
  <filter name="group_by_exit_day" string="Exit" context="{ 'group_by':
'exit_day:day' }"/>
</search>
```

La etiqueta *field* dentro de un *search* permite indicar por qué *fields* buscará. Se puede poner el atributo *filter_domain* si queremos incorporar una búsqueda más avanzada incluso con varios *fields*. Se usará la sintaxis de los dominios que ya hemos usado en *Odoo* en otras ocasiones.

La etiqueta *filter* establece un filtro predefinido que se aplicará pulsando en el menú. Necesita el atributo *domain* para que haga la búsqueda.

La etiqueta *filter* también puede servir para agrupar en función de un criterio. Para ello, hay que poner en el *context* la clave *group_by*, de forma que hará una búsqueda y agrupará por el criterio que le digamos. Hay un tipo especial de agrupación por fecha (último ejemplo) en la que podemos especificar si queremos agrupar por día, mes u otros.

7 ACCIONES

Las acciones permiten definir el comportamiento del sistema en respuesta a acciones del usuario, como pinchar en un menú o pinchar en una fila de una lista. Al igual que las vistas, se almacenan en la base de datos y se definen mediante XML con la etiqueta *record* con dos atributos, el modelo (*model*) y el *id*.

- El modelo permite definir el tipo de acción, tipo que define cual es su objetivo. Es por lo tanto un atributo obligatorio.
- El *id* también es obligatorio y permite la identificación unívoca de la acción dentro de la base de datos. Se suele prefijar con el nombre del módulo para evitar colisiones con otros posible módulos.

Además de estos dos atributos es habitual que sea cual sea el tipo de acción tenga un *field* con el atributo *name* que incluya un descripción corta de la acción.

7.1 Windows action

La acción de tipo Window (*ir.actions.act_window*) es la más habitual. Su objetivo es la presentación de los datos de un modelo en un vista.

```
<record model="ir.actions.act_window" id="prueba.action_window">
  <field name="name">Prueba window</field>
  <!-- se quiere mostrar datos del modelo prueba.user -->
  <field name="res_model">prueba.user</field>
  <!-- se quiere mostrar en formato form (formulario)-->
  <field name="view_mode">form</field>
</record>
```

Como *fields* obligatorios requiere:

- Modelo que se quiere presentar. Para definirlo se utiliza el valor *res_model* en el atributo *name*.
- La vista o las vistas que se necesitan ver de ese modelo, vistas que se definen en el *field* con el atributo *name* con el valor *view_mode*.

Odoo asociará directamente la vista definida para ese modelo y ese tipo. En el caso de que no haya ninguna definida y la que se quiera mostrar sea de tipo *tree* o *form*, *Odoo* generará por defecto.

Si existirán más de una vista por tipo de modelo (por ejemplo, dos vistas de tipo *tree*) habría varias opciones para indicar la vista a mostrar:

- se elegiría la vista por defecto, ya sea por que se carga antes o porque tiene una prioridad más baja (*field priority* de la vista).
- se crea un record basado en el modelo *ir.actions.act_window.view* donde se indica el tipo de vista, el identificador de la vista que se quiere mostrar y a la acción con la que se vincula

```
<record model="ir.actions.act_window" id="prueba.action_window">
  <field name="name">Muestra usuarios</field>
  <field name="type">ir.actions.act_window</field>
  <field name="res_model">prueba.user</field>
  <field name="view_mode">tree</field>
</record>

<record model="ir.actions.act_window.view" id="prueba.action_window_tree">
  <field name="view_mode">tree</field>
  <field name="view_id" ref="prueba.list2"/>
  <field name="act_window_id" ref="prueba.action_window"/>
</record>

<record model="ir.ui.view" id="prueba.list2">
  <field name="name">prueba_list2</field>
  <field name="model">prueba.user</field>
  <field name="arch" type="xml">
    <tree>
      <field name="name"/>
      <field name="surname"/>
      <field name="year"/>
    </tree>
  </field>
</record>
```



La etiqueta *tree* (y por lo tanto la vista) incluye por defecto botones para crear y editar. En el caso de querer mostrar para un modelo de datos únicamente la vista *tree* (como en el ejemplo anterior) esos botones ofrecen una mala usabilidad ya que no funcionan. Para evitar mostrarlos hay que incluir en la etiqueta *tree* los atributos *create* y *edit* con valor *false*.

Es posible, y de hecho es lo más habitual, definir más de una forma de presentación del modelo. Para ello debemos incluir, separados por comas y sin espacios, los tipos de vistas que deseamos obtener en el *field view_mode*.

```
<record model="ir.actions.act_window" id="prueba.action_window">
  <field name="name">Prueba window</field>
  <!-- se quiere mostrar datos del modelo prueba.user -->
  <field name="res_model">prueba.user</field>
  <!-- se quiere mostrar en formato form (formulario) y tree (árbol) -->
  <field name="view_mode">tree,form</field>
</record>
```

8 BIBLIOGRAFIA

1. Documentación de Odoo

9 AUTORES

A continuación ofrecemos en orden alfabético (por apellido) el listado de autores que han hecho aportaciones a este documento.

- Jose Castillo Aliaga
- Sergi García Barea
- Alfredo Oltra