Sistemas de Gestión Empresarial

Anexo I
Web Controllers

Actualizado Febrero 2021

Licencia

Nomenclatura

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

📖 Importante

❕ Atención

💬 Interesante

ÍNDICE DE CONTENIDO

UD08. ANEXO I WEB CONTROLLERS

1. INTRODUCCIÓN

Odoo tiene, de forma oficial, 3 clientes web distintos: El Backend, el TPV y el Frontend. Los tres funcionan de forma independiente y con algunas diferencias. Para la gestión de la empresa normalmente es suficiente con el backend, pero para el TPV o la página web no siempre es fácil o conveniente usar la solución de Odoo.

Odoo proporciona una manera de conectarse a él por XML-RPC, la cual es bastante simple y funciona bien cuando se está conectando alguna aplicación hecha con PHP, Python o Java, por ejemplo. De hecho, en la documentación oficial hay ejemplos para estos 3 lenguajes. No obstante, Javascript tiene problemas con este tipo de conexión. No es imposible y existen librerías para hacerlo, pero javascript y los frameworks del mismo están mucho más preparados para consultar API REST, por ejemplo, que para utilizar XML-RPC.

Por otro lado, los formularios, listas o kanban de Odoo se pueden quedar limitados si queremos mostrar los datos de alguna manera determinada. Para ello existen varias soluciones, pero normalmente optaremos por hacer un nuevo Componente (o widget, en versiones anteriores a la 14) o por insertar un fragmento de HTML obtenido por un Web Controller. En este manual trataremos el segundo caso.

Como vemos, hay muchas opciones y casi siempre se pueden hacer las cosas de muchas formas. Además, la documentación oficial de Odoo a este respecto es muy escueta y en ocasiones obsoleta. Es por eso que elegir no siempre es fácil. Reflexionemos sobre qué podemos hacer y cómo:

- Queremos un cambio menor en la apariencia: Añadir reglas CSS nuevas en el directorio static.
- Una nueva forma de visualizar o editar los datos: Crear un Componente o Widget.
- Una nueva forma de visualizar o editar un registro entero: Crear una vista.
- Comunicar una página web hecha con PHP con Odoo: XML-RPC.
- Hacer una web: En general se recomienda usar el módulo de web de Odoo y modificarla.
- Hacer una app con Java que se conecta al servidor Odoo: Usar XML-RPC o hacer una API REST con los Web Controllers.
- Hacer una web estática desde cero: Usar los Web Controller para generar HTML a partir de plantillas QWeb.
- Hacer una SPA desde cero con Angular, por ejemplo: Usas los Web Controller para generar JSON a partir de los datos. Hacer una API REST con Web Controllers

Las dos últimas opciones son las que vamos a explorar en este anexo.

2. WEB CONTROLLERS

Se trata de funciones que responden a URIs concretas. El servidor Odoo tiene un sistema de rutas para atender las peticiones. Por ejemplo, cuando accedemos a /web lo que proporciona es el backend. Si accedemos a /pos/web lo que nos proporciona es el Point of Sale. Nosotros podemos crear nuestras propias rutas para obtener páginas web personalizadas o otros datos como XML o JSON.

Para hacer estas rutas usamos los Web Controllers, de hecho, cuando creamos un nuevo módulo

con scaffold, se crea un fichero controllers.py que, por defecto, está comentado.

Veamos un ejemplo mínimo de controlador para analizarlo:

```
class MyController(http.Controller):
       @http.route('/school/course/', auth='user', type='json')
       def course(self):
...
```

Lo primero que podemos ver es que la clase hereda de **http.controller**. Recordemos que los modelos heredan de models.Model, por ejemplo.

Esta clase le da las propiedades necesarias para atender peticiones HTTP y queda registrada como un controlador web. Esta clase tendrá como atributos unas funciones con un decorador específico. Estas funciones se ejecutarán cada vez que el usuario acceda a la ruta determinada en el decorador.

El decorador **@http.route** permite indicar:

- La ruta que atenderá
- El tipo de autenticación con **auth**=, que puede ser **users** si requiere que esté autenticado, **public** si puede no estarlo o **none** para no tener en cuenta para nada la autenticación o el usuario actual.
- El tipo de datos que espera recibir y enviar con **type**= que pueden ser **http** o **json**. Esto quiere decir que puede aceptar datos con la sintaxis HTTP de GET o POST o que espera a interpretar un body de JSON.
- Los métodos HTTP que acepta con **methods=**
- La manera en la que trata las peticiones Cross-origin con **cors**=
- Si necesita una autenticación con **csrf**=

Las dos últimas opciones se deben poner en caso de hacer peticiones por una aplicación externa. En ese caso se pondrá: **cors='*', csrf=False,** ya que queremos que acepte cualquier petición externa y que no autentique con csrf, ya que tendremos que implementar nuestra propia autenticación.

Esta función recibirá parámetros, como veremos en el siguiente apartado y retornará una respuesta. En caso de ser type=’json’ retornará un JSON y en caso de ser http, retornará preferiblemente un HTML. Dejemos el JSON para más adelante y centrémonos en cómo crear HTML.

Odoo tiene su motor de plantillas HTML que es QWeb, a partir de este se puede construir el HTML con datos que queramos. También podemos no usar el motor de plantillas y crear algorítmicamente el HTML desde python. Esta segunda opción sólo es recomendada si va a ser muy simple o estático.

Veamos el ejemplo que proporciona Odoo cuando hacemos scaffold para crear un módulo:

```
@http.route('/school/school/objects/', auth='public')
def list(self, **kw):
   return http.request.render('school.listing', {
       'root': '/school/school',
       'objects': http.request.env['school.school'].search([]),
   })
```

Aquí se utiliza una nueva herramienta que es **http.request.render()**. Esta función utiliza una plantilla para crear un HTML a partir de unos datos.

Veamos ahora la plantilla school.listing

```
<template id="school.listing">
       <ul>
              <li t-foreach="objects" t-as="object">
              <a t-attf-href="#{ root }/objects/#{ object.id }">
              <t t-esc="object.display_name"/>
              </a>
              </li>
       </ul>
</template>
```

En esta plantilla, objects es el recordset que recorrerá y mostrará una lista de los registros enviados por el request.render.

Como hemos comentado anteriormente, es posible generar el HTML desde Python, aunque no es tan recomendable.

Todo lo anterior funciona perfectamente si accedemos a esta ruta desde el mismo navegador en el que hemos hecho login anteriormente. De esta manera se cumplirá que no estamos haciendo una petición Cross-origin y que estamos autenticados. El caso de hacer una petición desde otro lugar lo exploraremos en el apartado del API REST. Pero es posible que queramos mostrar cosas a usuarios que no tenemos autenticados, como por ejemplo un catálogo en una web.

Ya tenemos auth='public', pero si no está autenticado, la función search va a fallar. Para que no falle se puede poner sudo() antes de la función. De esta manera ignora si el usuario tiene permisos o si está autenticado.

```
'objects': http.request.env['school.school'].sudo().search([]),
```

> ❗ Mucho cuidado con el uso de sudo(). En general es mejor confiar en la autenticación de Odoo para todo. Si no es posible, hay que establecer un sistema de autenticación adecuado. En caso de ser información totalmente pública hay que limitar el acceso de los usuarios a lo mínimo imprescindible.

3. PASAR PARÁMETROS AL WEB CONTROLLER

Los métodos decorados con **@http.route** aceptan parámetros por el body de POST, por el ? de GET o en la misma URL como en el caso de un servicio REST. Veamos las tres opciones:

Parámetros por el POST o GET:

En caso de ser type='http', espera un body de POST tradicional o un GET con ? y & algo como:

```
parameter=value&also=another
```

Si sabemos el nombre de los parámetros, los podemos poner en la función decorada. En caso de desconocerlos, podemos poner ****args** y de esa manera **args** será una lista de parámetros. No es preciso que sea args, de hecho, en el ejemplo ponen **kw.

Veamos un ejemplo:

```
@http.route('/school', auth='public',cors='*', type='http')
        def get_course(self, model, obj, **kw):
        model = http.request.env[model].sudo().search([('id','=',obj)])
                    .mapped(lambda p: p.read()[0])
        return model
```

Este ejemplo se puede llamar con un POST en el que enviemos model=course&obj=23 o con un GET en el que la URI sería: <url>:8069/school?model=course&obj=23.

Parámetros por REST:

Modifiquemos este ejemplo para aceptar peticiones al modo de los servicios REST:

```
@http.route('/school/<model>/<obj>', auth='public',cors='*', type='http')
        def get_course(self, model, obj, **kw):
        model = http.request.env[model].sudo().search([('id','=',obj)])
                    .mapped(lambda p: p.read()[0])
        return model
```

No hay más que hacer una petición POST o GET sin enviar nada pero con la URI: <url>/school/course/23

Parámetros por JSON:

Tan solo tenemos que cambiar type='http' por **type='json'** y enviar un POST. Odoo necesita que el POST tenga como cabecera **"Content-Type: application/json"** y el body tendrá esta sintaxis en el caso del ejemplo anterior:

```
'{"jsonrpc":"2.0","method":"call","params":{"model":"course","obj":"23"}}'
```

Para probar todas estas peticiones se recomienda usar programas como PostMan, que permiten hacer muchas pruebas rápidamente. En caso de hacer alguna prueba puntual, se puede usar curl con comandos como este:

```
curl -i -X POST -H "Content-Type: application/json" \
-d '{"jsonrpc":"2.0","method":"call","params":{"model":"course","obj":"23"}}' \
localhost:8069/school
```

El hecho de poner type='json' o http cambia mucho el comportamiento de la función decorada. En cuanto a la forma de aceptar los datos ha quedado claro, pero la forma de retornarlos también cambia. Cuando es de tipo http, le decimos que retorne un HTML, pero no sólo retorna eso, sino que modifica las cabeceras para indicar que es de tipo http. Si queremos aceptar datos por el método GET será más fácil decir que es de type='http', pero si queremos que retorne un JSON, hay que indicarlo manualmente manipulando el **http.Response:**

```
return http.Response(
   json.dumps(resp),
   status=200,
   mimetype='application/json'
```

```
)
```

El uso de http.response no es necesario en caso de retornar un JSON por un decorador de tipo json o un HTML por un decorador http. El propio decorador ya añade esas cabeceras y serializa los datos.

Hay un problema con el objeto que retornamos, ya que json.dumps no siempre podrá serializar todo tipo de datos. Puede que tengamos que transformarlos nosotros.

CORS CON ODOO

En caso de hacer peticiones desde otra URL, como en el caso de las API, hay que configurar el cors= a los clientes que acepta. Normalmente pondremos cors="*" como en algunos ejemplos anteriores.

Al hacer peticiones Cross-Origin, las cookies que envia el servidor al autenticar no se registran en el navegador, por lo que es necesario implementar un protocolo de sesión por Token.

Si queremos que todo Odoo acepte CORS, lo mejor es configurar un Nginx como proxy y ya de paso configurar el HTTPS.

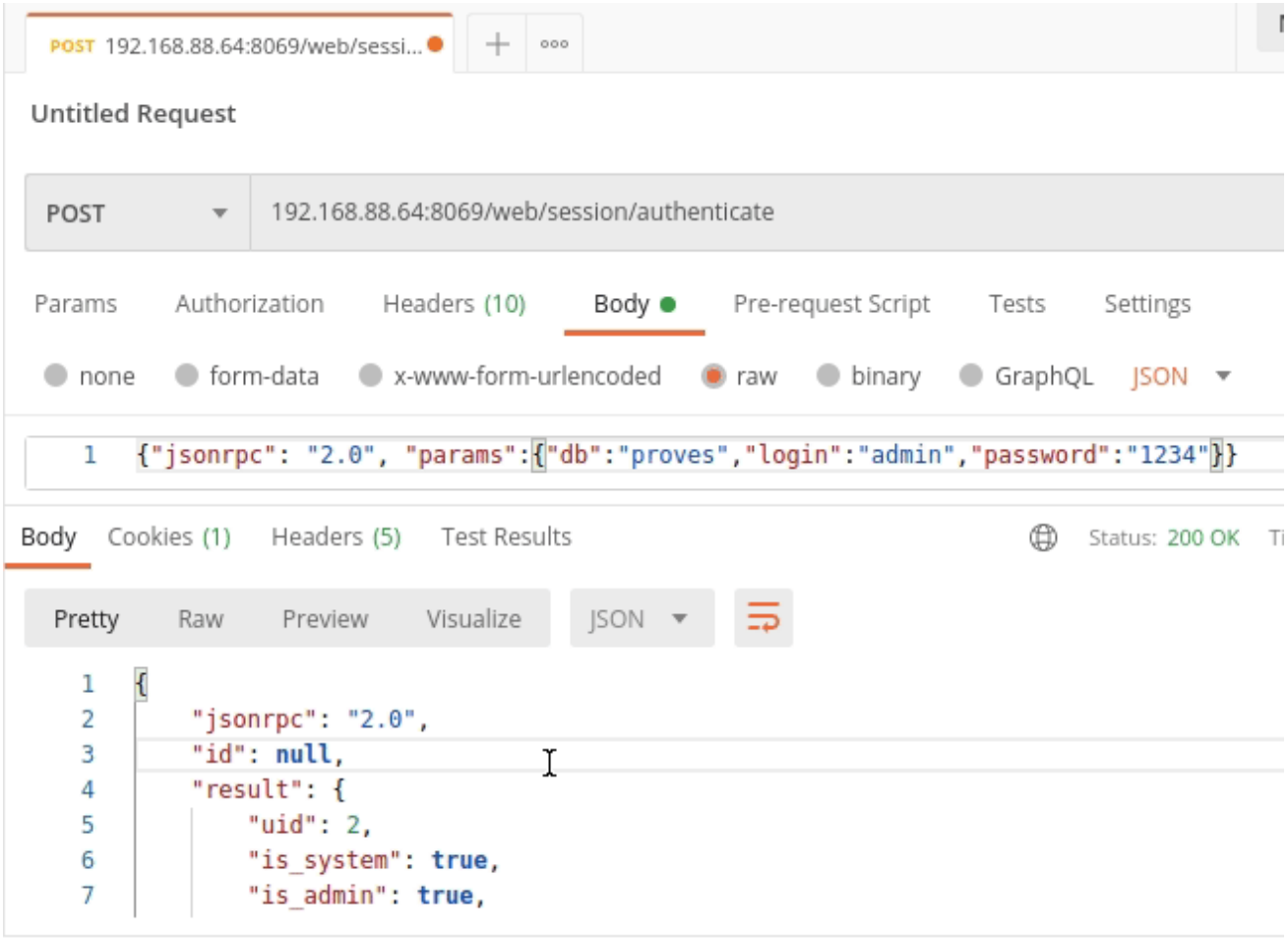
AUTENTICACIÓN

Este es el código del controlador de autenticación en Odoo:

```
@http.route('/web/session/authenticate', type='json', auth="none")
 def authenticate(self, db, login, password, base_location=None):
     request.session.authenticate(db, login, password)
     return request.env['ir.http'].session_info()
```

En caso de hacer una aplicación web en la misma URL y necesitar autenticación con un usuario de Odoo, tendremos que hacer una petición con JSON a esta ruta.

Aquí vemos un ejemplo con el programa PostMan:



HACER UNA API REST

Odoo está más orientado a crear webs con su framework o en su URL que para hacer de API. Pero si queremos hacer una aplicación móvil o una aplicación web externa que consulte sus datos, podemos optar por crear una API REST. Debemos tener en cuenta los siguientes factores:

- Hay que poner cors="*" para poder acceder.
- Debemos desactivar csrf, ya que no lo podemos usar.
- No podemos usar directamente la autenticación con Odoo. Necesitamos implementar algún protocolo para mantener la sesión, algo como Tokens JWT.
- En las API REST, el método de la petición es el verbo, así que hay que obtener el método para hacer cosas distintas.
- Si en el decorador ponemos type='json', no puede aceptar peticiones GET, ya que no tienen un body. Pero si ponemos type='http' hay que retornar un json igualmente. Depende de lo que pidamos por GET y cómo lo implementemos, podemos tener un problema al convertir de recordset a json con json.dumps(), pero lo podemos solventar con una herramienta interna de odoo sacada de la biblioteca **odoo.tools.date_utils**.

Veamos un ejemplo:

```
@http.route('/school/api/<model>', auth="none", cors='*', csrf=False,
methods=["POST","PUT","PATCH"] ,type='json')
def apiPost(self, **args):
  print('********** API POST PUT ************************************')
  print(args, http.request.httprequest.method)
  model = args['model']
  if (http.request.httprequest.method == 'POST'):
      record = http.request.env['school.' + model].sudo().create(args['data'])
      return record.read()
  if (http.request.httprequest.method == 'PUT' or http.request.httprequest.method == 'PATCH'):
     record = http.request.env['school.' + model].sudo().search([('id', '=', args['id'])])[0]
     record.write(args['data'])
     return record.read()
  return http.request.env['ir.http'].session_info()

@http.route('/school/api/<model>', auth="none", cors='*', csrf=False, methods=["GET", "DELETE"],
type='http')
def apiGet(self, **args):
  print('********** API GET DELETE ************************************')
  print(args, http.request.httprequest.method)
  model = args['model']
  search = []
  if 'id' in args:
      search = [('id', '=', args['id'])]
  if (http.request.httprequest.method == 'GET'):
      record = http.request.env['school.' + model].sudo().search(search)
      return http.Response( # Retornará un array sin el formato '{"jsonrpc":"2.0"...
       json.dumps(record.read(), default=date_utils.json_default),
       status=200,
       mimetype='application/json'
       )
  if (http.request.httprequest.method == 'DELETE'):
     record = http.request.env['school.' + model].sudo().search(search)[0]
     record.unlink()
     return http.Response(
       json.dumps(record.read(), default=date_utils.json_default),
       status=200,
       mimetype='application/json'
   )
  return http.request.env['ir.http'].session_info()
```

En este ejemplo falta todo lo relativo a la autenticación y algunas comprobaciones para evitar errores, pero se puede ver cómo hacemos una cosa distinta en función del método HTTP. Resulta más fácil de gestionar el GET y el POST por separado por el type.

COMUNICAR UNA SPA ANGULAR CON ODOO

Este apartado no tiene mucho que ver con el módulo, pero es interesante como enlace con DWEC de DAW o como introducción a un proyecto final de ciclo.

Este sería el servicio de Angular que hace peticiones al API REST del apartado anterior:

```
@Injectable({
  providedIn: 'root'
```

```
})

export class CourseService {
  courseURL = environment.url+'course';  // La URL está en enviroment
  postOptions = { headers: new HttpHeaders({ "Content-type": "application/json; charset=UTF-8"
})};
  constructor( private http: HttpClient) {   }
  createCourse(course:ICourse): Observable<ICourse[]> {
       let postBody =
`{"jsonrpc":"2.0","method":"call","params":{"data":"${JSON.parse(course)}"}}`;
       let obs: Observable<ICourse[]> =
       this.http.post<{result: ICourse[]}> (this.courseURL,this.postBody,this.postOptions)
           .pipe(map(response => response.result))
       return obs;
  }
```

4. BIBLIOGRAFÍA

- Sistemes de Gestió Empresarial IOC: https://ioc.xtec.cat/materials/FP/Materials/2252_DAM/DAM_2252_M10/web/html/index.html
- Wikipedia: https://es.wikipedia.org/wiki/Sistema_de_planificaci%C3%B3n_de_recursos_empresariales
- Documentación de Odoo: https://www.odoo.com/documentation/master/reference/http.html

5. AUTORES (EN ORDEN ALFABÉTICO)

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

- Jose Castillo Aliaga
- Sergi García Barea