Web контролери. Створення API. Конспект

1. Контролери, типи ендпоінтів HTTP, JSON, нововведення в 16 версії

Створення контролеру

Щоб створити власний контролер в Odoo, нам треба створити новий клас, що наслідує ***http.Controller*** з модуля ***odoo.http***, а методи з декоратором ***http.route*** дозволяють створити ендпоінти. Кастомні контролери можна наслідувати, але, на відміну від моделей, використовується звичайне, пайтонівське наслідування

```
from odoo import http
from odoo.http import request


class ProbePartnerController(http.Controller):

    @http.route(['/res_partner_probe_http/'], type='http', auth='public', )
    def res_partner_probe_http(self, **kwargs):
        res = request.env['res.partner'].sudo().search([])
        return ''.join([f'<div>{x.name}</div>' for x in res])

    @http.route(['/res_partner_probe_json/'], type='json', auth='public', )
    def res_partner_probe_json(self, **kwargs):
        data = request.env['res.partner'].sudo().search([])
        return data
```

Декоратор http.route

Розглянемо параметри декоратора http.route і на що вони впливають.

route

рядок або список роутів, які даний ендпоніт буде обробляти. Допускається використовувати іменовані параметри, які можуть бути типізованими.

```
@http.route(['/web/image',
    '/web/image/<string:xmlid>',
    '/web/image/<string:xmlid>/<string:filename>',
    '/web/image/<string:xmlid>/<int:width>x<int:height>',
    '/web/image/<string:xmlid>/<int:width>x<int:height>/<string:filename>',
    '/web/image/<string:model>/<int:id>/<string:field>',
    '/web/image/<string:model>/<int:id>/<string:field>/<string:filename>',
    '/web/image/<string:model>/<int:id>/<string:field>/<int:width>x<int:height>',

'/web/image/<string:model>/<int:id>/<string:field>/<int:width>x<int:height>/<string:fil
ename>',
    '/web/image/<int:id>',
    '/web/image/<int:id>/<string:filename>',
    '/web/image/<int:id>/<int:width>x<int:height>',
    '/web/image/<int:id>/<int:width>x<int:height>/<string:filename>',
    '/web/image/<int:id>-<string:unique>',
    '/web/image/<int:id>-<string:unique>/<string:filename>',
    '/web/image/<int:id>-<string:unique>/<int:width>x<int:height>',
    '/web/image/<int:id>-<string:unique>/<int:width>x<int:height>/<string:filename>'],
type='http', auth="public")
def content_image(self, xmlid=None, model='ir.attachment', id=None, field='raw',
                  filename_field='name', filename=None, mimetype=None, unique=False,
                  download=False, width=0, height=0, crop=False, access_token=None,
```

```
                nocache=False):
```

Параметри відповідно до імені додаються до параметрів, які передаються у метод-ендпоінт і їх можна прописати в параметри або, як і інші, забирати зі словника ***kwargs***.

type
- може бути ***'json'*** або ***'http'***.

auth
- може бути **'user'**, **'public'** або **'none'**. Перший потрібний якщо робити ендпоінт для вебсайту, для API не підходить. Третій - специфічний, для модулів, наприклад, авторизації, в цілому, для API не потрібний. **'public'** - це наш тип авторизації (він також є значення за замовчуванням). Можна робити ендпоінти для сайту, які не потребують авторизації.

csrf
- перевірка ***csrf***, для API `csrf=False` щоб не було помилки при запитах методом POST, для сторінок.

cors
- перевірка ***cors***. Якщо ваш API буде використовувати не браузер (мобільний додаток, сайт, інша система, головне не web застосунок), то ніяких проблем не буде.

Щоб обходити обмеження в браузері, потрібно додавати спеціальні заголовки у відповідь, наприклад

```
response.headers.set('Access-Control-Allow-Origin', '*')
response.headers.set(
   'Access-Control-Allow-Methods', 'GET, POST, PUT, OPTIONS')
response.headers.set(
   'Access-Control-Allow-Headers',
   'Origin, X-Requested-With, Content-Type, Accept, '
   'X-Debug-Mode, access-token, authorization')
```

Dispatcher
В Odoo є два стандартних типи контролерів це http та json. До 16 версії вони були єдиними можливими, в 16 з'явилась можливість додавати власні типи, шляхом наслідування від класу Dispatcher. Можна перевизначити будь-які параметри або методи, щоб зробити обробку ендпоінту більш зручною.

```
class KwApiDispatcher(Dispatcher):
   routing_type = 'kw_api'

   @classmethod
   def is_compatible_with(cls, request):
       return True

   def dispatch(self, endpoint, args):
       jsonrequest = json.loads(
           self.request.httprequest.get_data(as_text=True) or '{}')
       self.request.params = dict(jsonrequest.get('params', {}), **args)
       return endpoint(**self.request.params)

   def handle_error(self, exc):
       self.request.make_json_response({'error': exc})
```

2. HTTP ендпоінт, рендер шаблонів

Особливість

Ендпоінти типу ***http*** були призначені для видачі інформації користувачу, що зайшов на нього за допомогою веб браузеру. У версіях по 15 включно, odoo вимагає відсутності заголовку

Content-Type: application/json

Він може мати інше значення, аби не `'application/json', 'application/json-rpc'`.

http ендпоінт не має додаткових обгорток і будь-яке значення повернене через ***return*** буде конвертовано в рядок і відправлене користувачу, тобто ми можемо повертати не лише HTML, але й XML або JSON.

Рендер шаблонів

Рендер є зручним механізмом використання шаблонів odoo для відображення інформації. Тут слід зауважити, що механізм є зручним для розробника, але не є ефективним, щодо споживання ресурсів. Якщо інформації потрібно передати дуже багато, треба застосовувати інші механізми.

Розглянемо як викликати рендер шаблонів

```
@http.route(['/res_partner_probe_http_2/'], type='http', auth='public',
            website=True,)
def res_partner_probe_http(self, **kwargs):
    res = request.env['res.partner'].sudo().search([])
    return request.render(
        'probe_api.res_partner_template', {'partners': res})
```

Тут все доволі просто: ми вибираємо з БД потрібні нам дані і передаємо їх на рендерінг у метод request.render. Першим параметром є зовнішній ID шаблону, а другим словник з потрібними даними.
Сам шаблон є стандартним qweb шаблоном і може як використовувати інші шаблони, так і бути повністю самостійним.

```
<?xml version="1.0" encoding="utf-8"?>
<odoo>
    <template id="res_partner_template" name="Partners">
        <t t-call="website.layout">
            <div class="oe_structure">
                <div class="container">
                    <br/>
                    <center>
                        <h3>Partners</h3>
                    </center>
                    <br/>
                    <table class="table-striped table">
                        <thead style="font-size: 23px;">
                            <tr>
                                <th>Name</th>
                                <th>ID</th>
                            </tr>
                        </thead>
                        <tbody>
```

```
                        <t t-foreach="partners" t-as="partner">
                            <tr>
                                <td>
                                    <t t-esc="partner.name"/>
                                </td>
                                <td>
                                    <t t-esc="partner.id"/>
                                </td>
                            </tr>
                        </t>
                    </tbody>
                </table>
            </div>
        </div>
    </t>
</template>
</odoo>
```

Контролери website

Модуль ***website*** і похідні від нього, наприклад, ***website_sale*** додають нову функціональність в контролери.

website

визначає чи є ендпоінт частиною вебсайту odoo. Дозволяє отримувати сесію користувача, наприклад, відкритий sale order

```
order = request.website.sale_get_order()

а також інший контекст.
```

sitemap

впливає на те, чи буде даний ендпоінт сторінкою (і буде відображатись у sitemap.xml)

multilang

визначає чи буде ендпоінт прокидатися на локалізацію в залежності від локалі користувача. Можна використовувати для локалізації API.

Видача бінарних файлів

Є два способи віддавати формувати файли і віддавати їх в ендпоінт: файли з диску та файли з моделі ir.attachment

Видача файлу з диску

```
@http.route('/custom_report/<int:report_id>.xlsx', methods=['GET', ],
            auth='user', csrf=False, website=True, type='http', )
def custom_report_xlsx(self, report_id, **kw):
    custom_report = request.env['custom.report'].browse(report_id)
    file_name = custom_report.prepare_file()
    with open(file_name, 'rb') as file:
        headers = [
            ('Content-Disposition',
             f'attachment; filename="{custom_report.name}.xlsx"'),
            ('Content-Type',
```

```
            'application/vnd.openxmlformats-officedocument.'
            'spreadsheetml.sheet'),
           ('Content-Length', os.stat(file_name).st_size), ]
       return request.make_response(file.read(), headers=headers)
```

Важливо в заголовку правильно вказати тип, розмір файлу, можна вказати іншу назву файлу для зберігання у користувача. Метод ***request.make_response*** сформує правильну відповідь.

Приклад видачі файлу з атачментів

```
@http.route(route='/custom_attachment/<int:file_id>', methods=['GET'],
           auth='public', csrf=False, type='http', )
def custom_attachment(self, file_id, **kw):
   attachment = request.env['ir.attachment'].sudo().browse(file_id)
   res = Binary.content_common(
       request, id=attachment.id, filename=attachment.name,
       filename_field='name',
       access_token=attachment.generate_access_token()[0])
   res.headers['Content-Disposition'] = \
       'attachment; %s' % slugify_one(attachment.name)
   return res
```

в прикладі використовуємо контролер ***Binary***, який генерує заголовки і підготовлює бінарний файл для видачі

3. Робота з параметрами, завантаження файлів

Параметри

kwargs

Зазвичай потрібні параметри запиту доступні в параметрах метода-ендпоінта. Як в звичайних методах їх можна іменувати, а можна використовувати словник ****kwargs***. Ну і у разі, якщо іменований параметр не прийде, виникне помилка. Тому слід усім іменованим параметрам давати значення за замовчуванням. Перевагою використання ***kwargs*** є можливість перевіряти чи прийшов параметр.

Є певні особливості параметрів, які потрапляють до методу в залежності від його типу:
До http потрапляють дані аргументів запиту та полів форм, і не потрапляють дані тіла запиту
До json відповідно потрапляють дані тіла, а аргументи не потрапляють

request.httprequest.data

Містить тіло запиту. Саме він є джерелом даних json

request.httprequest.args

Містить значення аргументів запиту, зазвичай використовується в GET запитах.

request.httprequest.form

Значення полів форми. Зазвичай приходять в POST запитах.

request.httprequest.files

Містить значення прикріплених у форму файлів.

```
@http.route(['/res_partner_probe_http'], type='http', auth='public',
            website=True, multilang=False, csrf=False, )
def res_partner_probe_http(self, **kwargs):
    _logger.info('kwargs')
    _logger.info(kwargs)
    _logger.info('request.httprequest.data')
    _logger.info(request.httprequest.data)
    _logger.info('request.httprequest.args')
    _logger.info(request.httprequest.args)
    _logger.info('request.httprequest.form')
    _logger.info(request.httprequest.form)
    _logger.info('request.httprequest.files')
    _logger.info(request.httprequest.files)
    res = request.env['res.partner'].sudo().search([])
    return ''.join([f'<div>{x.name}</div>' for x in res])

@http.route(['/res_partner_probe_json'], type='json', auth='public',
            website=True, multilang=False, csrf=False, )
def res_partner_probe_json(self, **kwargs):
    _logger.info('kwargs')
    _logger.info(kwargs)
    _logger.info('request.httprequest.data')
    _logger.info(request.httprequest.data)
    _logger.info('request.httprequest.args')
    _logger.info(request.httprequest.args)
    _logger.info('request.httprequest.form')
    _logger.info(request.httprequest.form)
    _logger.info('request.httprequest.files')
    _logger.info(request.httprequest.files)
    data = request.env['res.partner'].sudo().search([])
    return data
```

Завантаження файлів в odoo

```
@http.route(route='custom_attachment', methods=['POST'],
            auth='public', csrf=False, type='http', )
def post_file(self, **kw):
    file = kw.get_param_by_name(kw, 'file')
    name = str(file.filename)
    file = base64.b64encode(file.read())
    attachment = request.env['ir.attachment'].sudo().create({
        'name': name, 'datas': file})
    file_url = request.httprequest.host_url[:-1] + attachment.local_url
    return file_url
```

Завантажений файл зберігається в тимчасовій локації і буде видалений після закриття обробки, тому важливо його відразу зберегти. Особливістю є необхідність конвертувати файл в base64 при записі в атачменти

4. Особливості роботи з великими файлами

Якщо дані або файл мають великий розмір, його видача може обірватись. Така ситуація може виникати вже на розмірах 50Мб.

```
from io import BytesIO
from werkzeug.wsgi import wrap_file

@http.route('/custom_report/<int:report_id>.xlsx', methods=['GET', ],
            auth='user', csrf=False, website=True, type='http', )
def custom_report_xlsx(self, report_id, **kw):
    custom_report = request.env['custom.report'].browse(report_id)
    file_name = custom_report.prepare_file()
    with open(file_name, 'rb') as file:
        buf = BytesIO(file.read())
        data = wrap_file(http.request.httprequest.environ, buf)

        headers = [
            ('Content-Disposition',
             f'attachment; filename="{custom_report.name}.xlsx"'),
            ('Content-Type',
             'application/vnd.openxmlformats-officedocument.'
             'spreadsheetml.sheet'),
            ('Content-Length', os.stat(file_name).st_size), ]
        response = http.Response(
            data, headers=headers, direct_passthrough=True)
        return response
```

Використання BytesIO разом з wrap_file забезпечує гарантовану закачку файлу.

5. Особливості JSON контролера в Odoo

В odoo json контролер призначений для обробки json-rpc запитів. По 15 версію включно наявність заголовку

Content-Type: application/json

перемикає обробку на json контролер, який має наступні особливості

- Вимагає наявності json даних
- Обов'язковість передачі json даних обмежує методи лише до POST запитів
- Ігнорує аргументи запиту (можна звернутись лише через `request.httprequest.args`)
- Не може отримувати файли
- Завжди огортає відповідь у json такого формату

```
{
   "jsonrpc": "2.0",
   "id": null,
   "result": "res.partner(3, 1)"
}
```

6. Приклад кастомного диспетчера в 16 версії

```
import json
import logging

from odoo import http
from odoo.http import request, Dispatcher, Response
```

```
_logger = logging.getLogger(__name__)


class CustomDispatcher(Dispatcher):
    routing_type = 'custom'

    @classmethod
    def is_compatible_with(cls, request):
        return True

    def handle_error(self, exc):
        self.request.make_json_response({'error': exc})

    def dispatch(self, endpoint, args):
        jsonrequest = json.loads(
            self.request.httprequest.get_data(as_text=True) or '{}')
        self.request.params = jsonrequest
        self.request.params.update(self.request.httprequest.args)
        body = endpoint(**self.request.params)
        body = json.dumps(body)
        response = Response(body, status=200, headers=[
            ('Content-Type', 'application/json'),
            ('Content-Length', len(body))])
        return response


class CustomController(http.Controller):

    @http.route(['/custom_custom_json'], type='custom', auth='public',
                website=True, multilang=False, csrf=False, )
    def custom_custom_json(self, **kwargs):
        data = request.env['res.partner'].sudo().search([]).mapped('name')
        return data
```

Розглянемо методи, які маємо перевизначити.

is_compatible_with

В цьому методі можна обмежити використання ендпоінту лише для окремих типів даних. В загальному випадку, він має виглядати як `return True`

handle_error

Даний метод призначений щоб описати обробку помилок. Можна повертати текст помилки у відповідь або логувати помилку.

dispatch

Цей метод описує дії які виконані перед переходом в ендпоінт, а також може обролювати результат, який віддає ендпоінт. Наприклад, встановити параметри, додати заголовки, огорнути в json тощо.

7. Створення контролера API за допомогою фреймворку kw_api

Знайомство

kw_api був створений для створення API для старіших версій та обійти обмеження, які були закладені в базову odoo. Для 16 версії вже немає необхідності використовувати манкіпатчі, щоб обійти обмеження, але інші можливості залишаються актуальними.

Розглянемо приклад

```
from odoo.addons.kw_api.controllers.controller_base import kw_api_route, \
   kw_api_wrapper, KwApi

from odoo import http


class KWController(http.Controller):

   @kw_api_route(route=['/kw_custom_custom', ], )
   @kw_api_wrapper(token=False, paginate=True, get_json=False, )
   def kw_custom_custom(self, kw_api, **kw):
       data = http.request.env['res.partner'].sudo().search([])
       return kw_api.data_response(data)
```

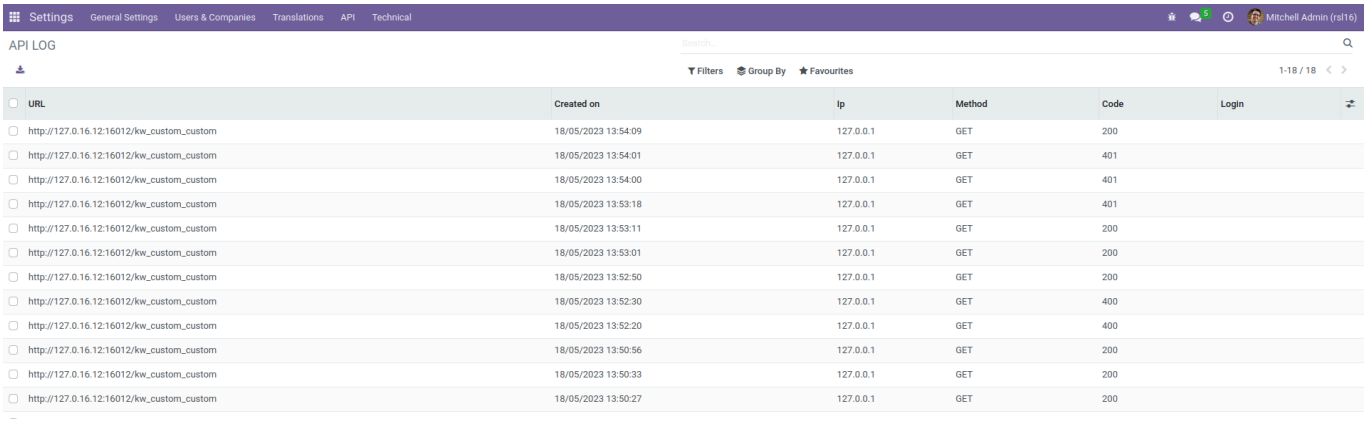
Результат має приблизно такий вигляд

```
{
   "content": [
       {
           "id": 3
       },
       {
           "id": 1
       }
   ],
   "totalElements": 2,
   "totalPages": 1,
   "numberOfElements": 2,
   "number": 0,
   "last": false
}
```

Можливості

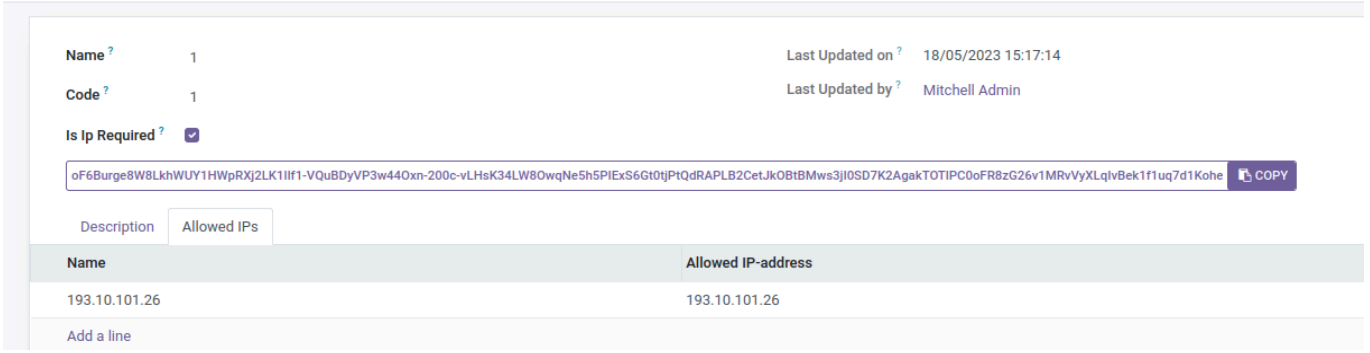
Логування

Усі запити, відповіді, помилки - все пишеться в логі, які можна перевірити в будь-який момент



Авторизація

Можливість налаштовувати авторизацію по токену (який оновлюється) або API ключу (який може обмежуватись по IP). На ендпонінті визначаються параметрами декоратора kw_api_wrapper token та api_key відповідно



Пагінація

Можливість автоматично відповідно до заданих або переданих користувачем значень розділяти результати на сторінки.

8. Створення API без програмування за допомогою kw_api_custom_endpoint

Додаткові матеріали

- **Web Controllers** - https://www.odoo.com/documentation/16.0/developer/reference/backend/http.html
- https://kw-api-doc.readthedocs.io/en/latest/
-

Додатково (можна додати):

- Наслідування контролерів

```
from odoo import http
from odoo.addons.website.controllers.main import Website

class WebsiteInfo(Website):

    @http.route()
    def website_info(self):
        result = super(WebsiteInfo, self).website_info()
        result.qcontext['apps'] = result.qcontext['apps'].filtered(lambda
x: x.name != 'website')
        return result
```

- Тестування контролерів
- Статичні файли

Всі файли, котрі розміщені за шляхом `/static` розглядаються як статичні ресурси, що доступні публічно. Ви можете розміщувати статичні дані будь-де у каталозі `static`, але рекомендується дотримуватися наступної структури, що залежить від типу файлу:

  - `/static/src/img` is the directory used for images.
  - `/static/src/css` is the directory used for CSS files.
  - `/static/src/scss` is the directory used for SCSS files.
  - `/static/src/fonts` is the directory used for font files.
  - `/static/src/js` is the directory used for JavaScript files.
  - `/static/src/xml` is the directory used for XML files for client-side QWeb templates.
  - `/static/lib` is the directory used for files of external libraries.

Наприклад, для доступу до зображення, що розміщено у каталозі `img` вашого модуля, використовуйте наступний URL: `/my_module/static/src/img/odoo.png`

- Використання GeoIP

```
country_code = request.session.geoip and \
    request.session.geoip.get('country_code') or 'UA'
```