# yezyilomo / odoo-rest-api

github.com/yezyilomo/odoo-rest-api

yezyilomo

## README.md

## Odoo REST API

This is a module which expose Odoo as a REST API

## Installing

- Download this module and put it to your Odoo addons directory
- Install requirements with `pip install -r requirements.txt`

## Getting Started

### Authenticating users

Before making any request make sure to login and obtain session_id(This will act as your Authentication token), Send all your requests with session_id as a parameter for authentication. There are two ways to obtain `session_id`, the first one is using `/web/session/authenticate/` route and the second one is using `/auth/` route.

- Using `/web/session/authenticate/` route

  Send a POST request with JSON body as shown below.

  `POST /web/session/authenticate/`

  Request Body

  ```
  {
      "jsonrpc": "2.0",
      "params": {
          "login": "your@email.com",
          "password": "your_password",
          "db": "database_name"
      }
  }
  ```

  Obtain `session_id` from a cookie created(Not the one from a response). It'll be a long string something like "62dd55784cb0b1f69c584f7dc1eea6f587e32570", Then you can use this as a parameter to all requests.

- Using `/auth/` route

  If you have set the default database then you can simply use `/auth` route to do authentication as

  `POST /auth/`

  Request Body

  ```
  {
      "params": {
          "login": "your@email.com",
          "password": "your_password",
          "db": "your_db_name"
      }
  }
  ```

  Use `session_id`

**Note:** For security reasons, in production don't send `session_id` as a parameter, use a cookie instead.

## Examples showing how to obtain `session_id` and use it

▶ Using `/web/session/authenticate/` route for authentication
▶ Using `/auth/` route for authentication
▶ Avoiding to send `session_id` as a parameter for security reasons

## Allowed HTTP methods

## 1. GET

### Model records:

`GET /api/{model}/`

#### Parameters

- **query (optional):**

  This parameter is used to dynamically select fields to include on a response. For example if we want to select `id` and `name` fields from `res.users` model here is how we would do it.

  `GET /api/res.users/?query={id, name}`

  ```
  {
      "count": 2,
      "prev": null,
      "current": 1,
      "next": null,
      "total_pages": 1,
      "result": [
          {
              "id": 2,
              "name": "Administrator"
           },
          {
              "id": 6,
              "name": "Reconwajenzi"
           }
       ]
   }
  ```

  For nested records, for example if we want to select `id` , `name` and `company_id` fields from `res.users` model, but under `company_id` we want to select `name` field only. here is how we would do it.

  `GET /api/res.users/?query={id, name, company_id{name}}`

  ```
  {
      "count": 2,
      "prev": null,
      "current": 1,
      "next": null,
      "total_pages": 1,
      "result": [
          {
              "id": 2,
              "name": "Administrator",
              "company_id": {
                  "name": "Singo Africa"
              }
           },
          {
              "id": 6,
              "name": "Reconwajenzi",
              "company_id": {
                  "name": "Singo Africa"
              }
           }
       ]
   }
  ```

For nested iterable records, for example if we want to select `id`, `name` and `related_products` fields from `product.template` model, but under `related_products` we want to select `name` field only. here is how we would do it.

`GET /api/product.template/?query={id, name, related_products{name}`

```
{
    "count": 2,
    "prev": null,
    "current": 1,
    "next": null,
    "total_pages": 1,
    "result": [
        {
            "id": 16,
            "name": "Alaf Resincot Steel Roof-16",
            "related_products": [
                {"name": "Alloy Steel AISI 4140 Bright Bars - All
5.8 meter longs"},
                {"name": "Test product"}
             ]
         },
         {
             "id": 18,
              "name": "Alaf Resincot Steel Roof-43",
              "related_products": [
                  {"name": "Alloy Steel AISI 4140 Bright Bars -
All 5.8 meter longs"},
                  {"name": "Aluminium Sheets & Plates"},
                  {"name": "Test product"}
              ]
         }
     ]
}
```

If you want to fetch all fields except few you can use exclude(-) operator. For example in the case above if we want to fetch all fields except `name` field, here is how we could do it
`GET /api/product.template/?query={-name}`

```
{
     "count": 3,
     "prev": null,
     "current": 1,
     "next": null,
     "total_pages": 1,
     "result": [
         {
             "id": 1,
             ... // All fields except name
         },
         {
             "id": 2
             ... // All fields except name
         }
         ...
     ]
}
```

There is also a wildcard(*) operator which can be used to fetch all fields, Below is an example which shows how you can fetch all product's fields but under `related_products` field get all fields except `id`.

`GET /api/product.template/?query={*, related_products{-id}}`

```
{
     "count": 3,
     "prev": null,
     "current": 1,
     "next": null,
     "total_pages": 1,
     "result": [
         {
             "id": 1,
             "name": "Pen",
             "related_products"{
                 "name": "Pencil",
                 ... // All fields except id
             }
             ... // All fields
         },
         ...
     ]
}
```

**If you don't specify query parameter all fields will be returned.**

- **filter (optional):**

  This is used to filter out data returned. For example if we want to get all products with id ranging from 60 to 70, here's how we would do it.

  `GET /api/product.template/?query={id, name}&filter=[["id", ">", 60], ["id", "<", 70]]`

```
{
    "count": 3,
    "prev": null,
    "current": 1,
    "next": null,
    "total_pages": 1,
    "result": [
        {
            "id": 67,
            "name": "Crown Paints Economy Superplus Emulsion"
        },
        {
            "id": 69,
            "name": "Crown Paints Permacote"
        }
    ]
}
```

- **page_size (optional) & page (optional):**

  These two allows us to do pagination. Hre page_size is used to specify number of records on a single page and page is used to specify the current page. For example if we want our page_size to be 5 records and we want to fetch data on page 3 here is how we would do it.

  `GET /api/product.template/?query={id, name}&page_size=5&page=3`

  ```
  {
      "count": 5,
      "prev": 2,
      "current": 3,
      "next": 4,
      "total_pages": 15,
      "result": [
          {"id": 141, "name": "Borewell Slotting Pipes"},
          {"id": 114, "name": "Bright Bars"},
          {"id": 128, "name": "Chain Link Fence"},
          {"id": 111, "name": "Cold Rolled Sheets - CRCA & GI
  Sheets"},
          {"id": 62, "name": "Crown Paints Acrylic Primer/Sealer
  Undercoat"}
      ]
  }
  ```

  Note: `prev` , `current` , `next` and `total_pages` shows the previous page, current page, next page and the total number of pages respectively.

- **limit (optional):**

  This is used to limit the number of results returned on a request regardless of pagination. For example

  `GET /api/product.template/?query={id, name}&limit=3`

  ```
  {
      "count": 3,
      "prev": null,
      "current": 1,
      "next": null,
      "total_pages": 1,
      "result": [
          {"id": 16, "name": "Alaf Resincot Steel Roof-16"},
          {"id": 18, "name": "Alaf Resincot Steel Roof-43"},
          {"id": 95, "name": "Alaf versatile steel roof"}
      ]
  }
  ```

## Model record:

`GET /api/{model}/{id}`

### Parameters

**query (optional):**

Here query parameter works exactly the same as explained before except it selects fields on a single record. For example

`GET /api/product.template/95/?query={id, name}`

```
{
    "id": 95,
    "name": "Alaf versatile steel roof"
}
```

## 2. POST

`POST /api/{model}/`

### Headers

Content-Type: application/json

### Parameters

- **data (mandatory):**

  This is used to pass data to be posted. For example

  `POST /api/product.public.category/`

  Request Body

  ```
  {
      "params": {
          "data": {
              "name": "Test category_2"
          }
      }
  }
  ```

  Response

  ```
  {
      "jsonrpc": "2.0",
      "id": null,
      "result": 398
  }
  ```

  The number on `result` is the `id` of the newly created record.

- **context (optional):**

  This is used to pass any context if it's needed when creating new record. The format of passing it is

  Request Body

  ```
  {
      "params": {
          "context": {
              "context_1": "context_1_value",
              "context_2": "context_2_value",
              ....
          },
          "data": {
              "field_1": "field_1_value",
              "field_2": "field_2_value",
              ....
          }
      }
  }
  ```

## 3. PUT

### Model records:

`PUT /api/{model}/`

#### Headers

Content-Type: application/json

#### Parameters

- **data (mandatory):**

  This is used to pass data to update, it works with filter parameter, See example below

- **filter (mandatory):**

  This is used to filter data to update. For example

  `PUT /api/product.template/`

  Request Body

  ```
  {
      "params": {
          "filter": [["id", "=", 95]],
          "data": {
              "name": "Test product"
          }
      }
  }
  ```

  Response

  ```
  {
      "jsonrpc": "2.0",
      "id": null,
      "result": true
  }
  ```

  Note: If the result is true it means success and if false or otherwise it means there was an error during update.

- **context (optional):** Just like in GET context is used to pass any context associated with record update. The format of passing it is

  Request Body

  ```
  {
      "params": {
          "context": {
              "context_1": "context_1_value",
              "context_2": "context_2_value",
              ....
          },
          "filter": [["id", "=", 95]],
          "data": {
              "field_1": "field_1_value",
              "field_2": "field_2_value",
              ....
          }
      }
  }
  ```

- **operation (optional)**:

  This is only applied to `one2many` and `many2many` fields. The concept is sometimes you might not want to replace all records on either `one2many` or `many2many` fields, instead you might want to add other records or remove some records, this is where put operations comes in place. Thre are basically three PUT operations which are push, pop and delete.

  - push is used to add/append other records to existing linked records
  - pop is used to remove/unlink some records from the record being updated but it doesn't delete them on the system
  - delete is used to remove/unlink and delete records permanently on the system

  For example here is how you would update `related_product_ids` which is `many2many` field with PUT operations

  `PUT /api/product.template/`

  Request Body

  ```
  {
      "params": {
          "filter": [["id", "=", 95]],
          "data": {
              "related_product_ids": {
                  "push": [102, 30],
                  "pop": [45],
                  "delete": [55]
              }
          }
      }
  }
  ```

  This will append product with ids 102 and 30 as related products to product with id 95 and from there unlink product with id 45 and again unlink product with id 55 and delete it from the system. So if befor this request product with id 95 had [20, 37, 45, 55] as related product ids, after this request it will be [20, 37, 102, 30].

  Note: You can use one operation or two or all three at a time depending on what you want to update on your field. If you dont use these operations on `one2many` and `many2many` fields, existing values will be replaced by new values passed, so you need to be very carefull on this part.

  Response:

```
{
    "jsonrpc": "2.0",
    "id": null,
    "result": true
}
```

### Model record:

`PUT /api/{model}/{id}`

#### Headers

Content-Type: application/json

#### Parameters

- data (mandatory)
- context (optional)
- PUT operation(push, pop, delete) (optional)

All parameters works the same as explained on previous section, what changes is that here they apply to a single record being updated and we don't have filter parameter because `id` of record to be updated is passed on URL as `{id}` . Example to give us an idea of how this works.

`PUT /api/product.template/95/`

Request Body

```
{
    "params": {
        "data": {
            "related_product_ids": {
                "push": [102, 30],
                "pop": [45],
                "delete": [55]
            }
        }
    }
}
```

## 4. DELETE

### Model records:

`DELETE /api/{model}/`

#### Parameters

**filter (mandatory):**

This is used to filter data to delete. For example

`DELETE /api/product.template/?filter=[["id", "=", 95]]`

Response

```
{
    "result": true
}
```

Note: If the result is true it means success and if false or otherwise it means there was an error during deletion.

### Model records:

`DELETE /api/{model}/{id}`

#### Parameters

This takes no parameter and we don't have filter parameter because `id` of record to be deleted is passed on URL as `{id}`. Example to give us an idea of how this works.

`DELETE /api/product.template/95/`

Response

```
{
    "result": true
}
```

## Calling Model's Function

Sometimes you might need to call model's function or a function bound to a record, inorder to do so, send a `POST` request with a body containing arguments(args) and keyword arguments(kwargs) required by the function you want to call.

Below is how you can call model's function

`POST /object/{model}/{function name}`

Request Body

```
{
    "params": {
        "args": [arg1, arg2, ..],
        "kwargs ": {
            "key1": "value1",
            "key2": "value2",
            ...
        }
    }
}
```

And below is how you can call a function bound to a record

`POST /object/{model}/{record_id}/{function name}`

Request Body

```
{
    "params": {
        "args": [arg1, arg2, ..],
        "kwargs ": {
            "key1": "value1",
            "key2": "value2",
            ...
        }
    }
}
```

In both cases the response will be the result returned by the function called