Connecting to Odoo from Standalone OWL App

Introduction

In order to create a functional standalone OWL app that runs on data from an Odoo instance even if it's not running on the same system as the app itself we can go at this in 1 of 3 ways

1. Connect using **RPC** 1.1. Connect using **XMLRPC** 1.2. Connect using **JSONRPC**
2. Connect using **Web Controllers**

1. Connecting using RPC

RPC stands for Remote Procedure Call and is a way to call methods on process instances from another process

It may be the easiest to call and execute but it's plagued with slow response times and security issues. It's also not the most efficient way to connect to Odoo

RPC is mostly used for simple, single command queries that require small amounts of data

1.1. Connecting using XMLRPC *(odoo-react-native)*

XMLRPC can be initialized straight from Javascript using the `odoo-react-native` library

```
import Odoo from "odoo-react-native";
```

Creating connector object

> 1. Creates a new Odoo object.
> 2. Sets the host, port, database, username, and password.
> 3. Creates a new connection to the Odoo server.
> 4. Returns the connection.

```
import Odoo from "react-native-odoo";

export const odooAPI = new Odoo({
  host: "localhost",
  port: 8069,
  database: "Demo",
  username: "admin",
  password: "admin",
});
```

Using the connection

> 1. Connect to Odoo.
> 2. Wait for 10 seconds for the connection to be established.
> 3. If the connection is established, get the partners.
> 4. If there is an error, log it.
> 5. Else if the connection is established, log the partners

```
import { odooAPI } from "./odoo";

export const nams = () => {
  //cursor
  // Connect to Odoo
  var connected = false;
  var intr = setInterval(function () {
    odooAPI.connect(function (err) {
      if (err) {
        return console.log(err);
      }
      connected = true;
    });
    // clear the interval if `i` reached 100
    if (connected == true) clearInterval(intr);
  }, 10000);

  var params = {
    ids: [1, 2, 3, 4, 5],
    fields: ["name"],
  }; //params
  odooAPI.get("res.users", params, function (err, partners) {
    if (err) {
      return console.log(err);
    }

    console.log(partners);
    // [
    //   { id: 1, name: 'Demo Company' },
    //   { id: 3, name: 'Administrator' },
    //   { id: 4, name: 'Public user' },
    //   { id: 5, name: 'Demo User' }
    // ]
  }); //get
};
```

1.2. Connecting using JSONRPC *(Python Odoorpc)*

JSONRPC is a way to call methods on process instances from another process using python in this case using the `odoorpc` library

```
import odoorpc
```

Creating connector object

1. Connect to the server at port `8069` with the `database` `"Demo"`
2. Login with the `user` `"Demo"` and the `password` `"admin"`

```
import flask
import odoorpc
from flask_cors import CORS

app = flask.Flask(__name__)
CORS(app)


# Prepare the connection to the server
odoo = odoorpc.ODOO("localhost", port=8069)

odoo.login("Demo", "admin", "admin")
```

Using the connection

1. Creates a **Flask** application called `app`.
2. Creates a **route** called `/test`.
3. If the request method is `GET`, returns a message with the current user's `name` and `ID`.
4. If the request method is `POST`, prints the request body as `JSON` and returns a success message.
5. Runs the application.

```
# Current user
user = odoo.env.user

@app.route("/test", methods=["GET", "POST"])
def testfn():
    # GET request
    if flask.request.method == "GET":
        message = {"data": user.name,
        "uid": user.id}
        return message  # serialize and use JSON headers
    # POST request
    if flask.request.method == "POST":
        print(flask.request.get_json())  # parse as JSON
        return "Sucesss", 200

app.run(debug=True)
```

2. Connecting using Web Controllers

Web Controllers are classes in Odoo that are able to catch the http requests sent by any browser. They allow to generate html pages to be served like any web server, implement new methods to be used by the Javascript client, etc

Controllers File By convention the controllers should be placed in the controllers directory of the module

Example

```
module_name
----controllers
--------__init__.py
--------my_controllers.py
----__init__.py
----__manifest__.py
```

The way Odoo web controllers work is by hosting the data on separate callable routes that can accept GET and POST requests externally in a way similar to a Flask server

This is a more efficient way to connect to Odoo Web Controller

- In `__init__.py` you must add:

```
from . import controllers
```

- Here is the content of `controllers/__init__.py`:

```
import my_controllers
```

- Now you can put the following content in `controllers/my_controllers.py`:

```
import openerp.http as http
from openerp.http import request
Controller Declaration...
```

In your `controllers` file, you can now declare a controller this way:

```
class MyController(http.Controller):

    @http.route('/my_url/some_html', type="http")
    def some_html(self):
```

```
        return "<h1>This is a test</h1>"

    @http.route('/my_url/some_json', type="json")
    def some_json(self):
        return {"sample_dictionary": "This is a sample JSON dictionary"}
```

Note: A controller must inherit from `http.Controller`. Each time you define a method with `@http.route()` it defines a url to match. As example, the `some_html()` method will be called a client query the `/my_url/some_html` url.

Fetching the Data from JavaScript

```
fetch("http://localhost:8069/my_url/some_html")
  .then(function (response) {
    return response.json();
  })
  .then(function (text) {
    console.log("GET response:");
    console.log(text);
  });
```

Reading Data From Odoo

- In `controllers/my_controllers.py` you can modify the method to call the `env` so it can return data from Odoo.

```
    thing = http.request.env[module_name].modifier().method(params);
```

Odoo's `env` supports a variety of modifiers to alter the behavior of methods getting data from odoo, including

- `filtered()`: filters the data returned by the method
- `sudo`: runs the method with the superuser's credentials
- `with_context()`: runs the method with the given context

Odoo's `env` supports a variety of methods to get data from Odoo, including

- `env.search(params)`: Search for records in Odoo

- `params` is a dictionary with the following keys used to filter the search:
- `domain`: A list of domain expressions
- `fields`: A list of fields to return in the result
- `limit`: Maximum number of records to return
- `offset`: Number of records to skip in the result

> - `order`: List of fields to order the result by
> - `context`: Context to use when searching for records

- `env.search_count(params)`: Count the number of records in Odoo

> - `params` is a dictionary with the following keys used to filter the search:
> - `domain`: A list of domain expressions
> - `context`: Context to use when searching for records

- `env.browse(ids)`: Get the records with the given ids

> - `ids` is a list of ids to get the records from Odoo

- `env.create(params)`: Create a record in Odoo

> - `params` is a dictionary with the following keys used to create the record:
> - `values`: A dictionary with the values to create the record with

- `env.write(params)`: Update a record in Odoo

> - `params` is a dictionary with the following keys used to update the record:
> - `id`: The id of the record to update
> - `values`: A dictionary with the values to update the record with

- `env.unlink(params)`: Delete a record in Odoo

> - `params` is a dictionary with the following keys used to delete the record:
> - `id`: The id of the record to delete

- `env.exists()`: Check if a record exists in Odoo - `env.copy()`: Copy a record in Odoo - `env.default_get()`: Get the default values of a record in Odoo - `env.name_get()`: Get the names of the records in Odoo - `env.name_search()`: Search for records in Odoo

> - `params` is a dictionary with the following keys used to search for records:
> - `name`: The name to search for
> - `args`: A list of domain expressions
> - `operator`: The operator to use for the search
> - `limit`: Maximum number of records to return

JSON Format for Data Transfer

Note: As the data is being passed through an http or a json Get and POST call, the data must be in JSON format to be transferred and can be created manually or using a library to do so.

This results in the method that creates the JSON format Data to be passed looking as follows

```
class MyController(http.Controller):
    @http.route("/my_url/some_html", type="http")
    def some_html(self):
        # _logger.info('FYI: This is happening')
```

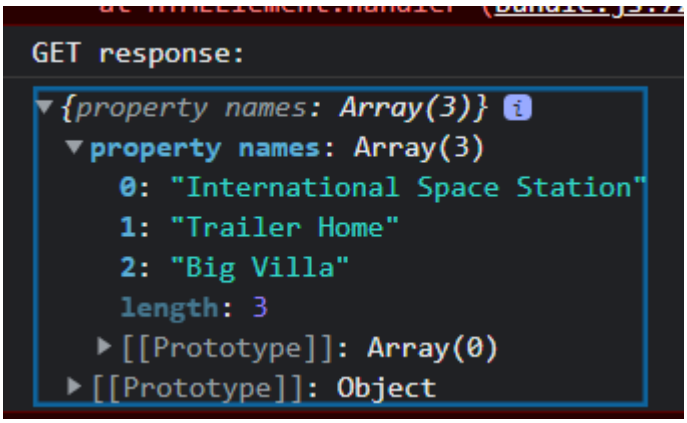
```
        jason="{\"property names\" : [";
        # _logger.warning('WARNING: I dont think you want this to happen!')

        # _logger.error('ERROR: Something really bad happened!')
        estate_properties = http.request.env['estate.properties']
        property_ids=estate_properties.search([])
        for propertee in property_ids:
            _logger.info(propertee.name)
            if (jason != "{\"property names\" : ["):
                jason+=", ";
            jason+=("\""+propertee.name+ "\"")
            _logger.info(jason)
        jason+="]}";
        json.dumps(jason)
        _logger.info(property_ids)
        _logger.info(jason)
        return jason

    @http.route("/my_url/some_json", type="json")
    def some_json(self):
        jason = json.dumps ({"sample_dictionary" : "This is a sample JSON
dictionary"})
        return jason
```

This method will return the following JSON format Data

```
{
    "property names" : [
        "Property 1",
        "Property 2",
        "Property 3"
    ]
}
```



This documentation was created by Ali Dandan with help of Github's Copilot