FREE eBook

LEARNING odoo-8

Free unaffiliated eBook created from
Stack Overflow contributors.

#odoo-8

Table of Contents

About

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: odoo-8

It is an unofficial and free odoo-8 ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official odoo-8.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Chapter 1: Getting started with odoo-8

Remarks

This section provides an overview of what odoo-8 is, and why a developer might want to use it.

It should also mention any large subjects within odoo-8, and link out to the related topics. Since the Documentation for odoo-8 is new, you may need to create initial versions of those related topics.

Versions

Release Number	Community	Enterprise	License	Release Date
8.0	Yes	No	GNU AGPL	2014-09-18
9.0	Yes	Yes	GNU AGPL V3	2015-10-01

Examples

Setup

Odoo can be installed in three different ways:

1. Packaged installers (easiest, less flexible)
2. Source install (takes sometime to setup, very flexible)
3. An official docker image from docker.com

Official packages with all relevant dependency requirements are available on odoo.com.

Windows

Download and run the installer.

Note: On Windows 8 you may see a warning titled "Windows protected your PC". Click More Info then Run it anyway. Accept the UAC prompt and go through the various installation steps. Odoo will automatically be started at the end of the installation.

Configuration:

The configuration file can be found at %PROGRAMFILES%\Odoo 8.0-id\server\openerp-server.conf. (id is your system username)

The configuration file can be edited to connect to a remote Postgresql, edit file locations or set a dbfilter. To reload the configuration file, restart the Odoo service via Services ▸ odoo server.

Linux

Debian based distributions

To install Odoo 8.0 on Debian-based distribution, execute the following commands as root:

```
# wget -O - https://nightly.odoo.com/odoo.key | apt-key add -
# echo "deb http://nightly.odoo.com/8.0/nightly/deb/ ./" >> /etc/apt/sources.list
# apt-get update && apt-get install odoo
```

This will automatically install all dependencies, install Odoo itself as a daemon and automatically start it.

Note that

to print PDF reports, you must install wkhtmltopdf yourself: the version of wkhtmltopdf available in debian repositories does not support headers and footers so it can not be installed automatically. The recommended version is 0.12.1 and is available on the wkhtmltopdf download page, in the archive section. As there is no official release for Debian Jessie, you can find the package on http://nightly.odoo.com/extra/. or you can download and install it from wkhtmltopdf's download page like this

```
# wget https://bitbucket.org/wkhtmltopdf/wkhtmltopdf/downloads/{path to correct distro and
system architecture}
# sudo dpkg -i {.deb package}
# sudo cp /usr/local/bin/wkhtmlto* /usr/bin/
```

The configuration file can be found at /etc/odoo/openerp-server.conf

When the configuration file is edited, Odoo must be restarted using service:

$ sudo service odoo restart Restarting odoo: ok

RPM based distributions

With RHEL-based distributions (RHEL, CentOS, Scientific Linux), EPEL must be added to the distribution's repositories for all of Odoo's dependencies to be available. For CentOS:

```
$ sudo yum install -y epel-release
```

For other RHEL-based distribution, see the EPEL documentation.

Below are the installation steps.

```
$ sudo yum install -y postgresql-server
$ sudo postgresql-setup initdb
$ sudo systemctl enable postgresql
$ sudo systemctl start postgresql
$ sudo yum-config-manager --add-repo=https://nightly.odoo.com/8.0/nightly/rpm/odoo.repo
$ sudo yum install -y odoo
$ sudo systemctl enable odoo
```

```
$ sudo systemctl start odoo
```

Note that

To print PDF reports, you must install wkhtmltopdf yourself: the version of wkhtmltopdf available in Fedora/CentOS repositories does not support headers and footers so it can not be installed automatically. Use the version available on the wkhtmltopdf download page. Configuration, similar to debian it can be installed with

```
wget https://bitbucket.org/wkhtmltopdf/wkhtmltopdf/downloads/{path to correct distro and
system architecture}
sudo rpm -i  {.rpm package}
sudo cp /usr/local/bin/wkhtmlto* /usr/bin/
```

The configuration file can be found at /etc/odoo/openerp-server.conf

When the configuration file is edited, Odoo must be restarted via Systemd:

```
$ sudo systemctl restart odoo
```

Source Install

Odoo zip can be downloaded from https://nightly.odoo.com/8.0/nightly/src/odoo_8.0.latest.zip, the zip file then needs to be uncompressed to use its content

Git allows simpler update and easier switching between differents versions of Odoo. It also simplifies maintaining non-module patches and contributions. The primary drawback of git is that it is significantly larger than a tarball as it contains the entire history of the Odoo project.

The git repository is `https://github.com/odoo/odoo.git`.

Then you can clone the repository with

```
$ git clone https://github.com/odoo/odoo.git
```

Installing dependencies

Source installation requires manually installing dependencies:

Python 2.7. on Linux and OS X, included by default

on Windows, use the official Python 2.7.9 installer.

if Python is already installed, make sure it is 2.7.9, previous versions are less convenient and 3.x versions are not compatible with Odoo

configuring PostgreSQL

After installation you will need to create a postgres user: by default the only user is postgres, and Odoo forbids connecting as postgres.

on Linux, use your distribution's package, then create a postgres user named like your login:

```
$ sudo su - postgres -c "createuser -s $USER"
```

Because the role login is the same as your unix login unix sockets can be use without a password. on OS X, postgres.app is the simplest way to get started, then create a postgres user as on Linux

on Windows, use PostgreSQL for windows then add PostgreSQL's bin directory (default: C:\Program Files\PostgreSQL\9.4\bin) to your PATH

create a postgres user with a password using the pg admin gui: open pgAdminIII, double-click the server to create a connection, select Edit ‣ New Object ‣ New Login Role, enter the username in the Role Name field (e.g. odoo), then open the Definition tab and enter the password (e.g. odoo), then click OK.

The user and password must be passed to Odoo using either the -w and -r options or the configuration file

Python dependencies listed in the requirements.txt file.

on Linux, python dependencies may be installable with the system's package manager or using pip.

For libraries using native code (Pillow, lxml, greenlet, gevent, psycopg2, ldap) it may be necessary to install development tools and native dependencies before pip is able to install the dependencies themselves. These are available in -dev or -devel packages for Python, Postgres, libxml2, libxslt, libevent, libsasl2 and libldap2. Then the Python dependecies can themselves be installed:

```
$ pip install -r requirements.txt
```

On OS X, you will need to install the Command Line Tools (xcode-select --install) then download and install a package manager of your choice (homebrew, macports) to install non-Python dependencies. pip can then be used to install the Python dependencies as on Linux:

```
$ pip install -r requirements.txt
```

on Windows you need to install some of the dependencies manually, tweak the requirements.txt file, then run pip to install the remaning ones.

```
Install psycopg using the installer here http://www.stickpeople.com/projects/python/win-
psycopg/
```

Then edit the requirements.txt file: remove psycopg2 as you already have it. remove the optional python-ldap, gevent and psutil because they require compilation. add pypiwin32 because it's needed under windows.

Then use pip to install the dependencies using the following command from a cmd.exe prompt (replace \YourOdooPath by the actual path where you downloaded Odoo):

```
C:\> cd \YourOdooPath
C:\YourOdooPath> C:\Python27\Scripts\pip.exe install -r requirements.txt
```

Less CSS via nodejs

on Linux, use your distribution's package manager to install nodejs and npm.

Note that

In debian wheezy and Ubuntu 13.10 and before you need to install nodejs manually:

```
$ wget -qO- https://deb.nodesource.com/setup | bash -
$ apt-get install -y nodejs
```

In later debian versions (>jessie) and ubuntu (>14.04) you may need to add a symlink as npm packages call node but debian calls the binary nodejs

```
$ apt-get install -y npm
$ sudo ln -s /usr/bin/nodejs /usr/bin/node

Once npm is installed, use it to install less and less-plugin-clean-css:

$ sudo npm install -g less less-plugin-clean-css

on OS X, install nodejs via your preferred package manager (homebrew, macports) then install
less and less-plugin-clean-css:

$ sudo npm install -g less less-plugin-clean-css
```

on Windows, install `nodejs`, reboot (to update the PATH) and install `less` and `less-plugin-clean-css`:

```
C:\> npm install -g less less-plugin-clean-css
```

Running Odoo

Once all dependencies are set up, Odoo can be launched by running odoo.py.

Configuration can be provided either through command-line arguments or through a configuration file.

Common necessary configurations are:

```
PostgreSQL host, port, user and password.
```

Odoo has no defaults beyond psycopg2's defaults: connects over a UNIX socket on port 5432 with the current user and no password. By default this should work on Linux and OS X, but it will not work on windows as it does not support UNIX sockets. Custom addons path beyond the defaults, to load your own modules

Under Windows a typical way to execute odoo would be:

```
C:\YourOdooPath> python odoo.py -w odoo -r odoo --addons-path=addons,../mymodules --db-filter=mydb$
```

Where odoo, odoo are the postgresql login and password, ../mymodules a directory with additional addons and mydb the default db to serve on localhost:8069

Under *nix systems a typical way to execute odoo would be:

```
$ ./odoo.py --addons-path=addons,../mymodules --db-filter=mydb$Packaged installers
```

What is Odoo?

Odoo (formerly known as OpenERP and before that, TinyERP) is a suite of open core enterprise management applications. Targeting companies of all sizes, the application suite covers all business needs, from Website/Ecommerce down to manufacturing, inventory and accounting, all seamlessly integrated. It is the first time ever a software editor managed to reach such a functional coverage. Odoo is the most installed business software in the world. Odoo is used by 2,000,000+ users worldwide ranging from very small companies (1 user) to very large ones (300,000 users).

The source code for the OpenObject framework and core ERP (enterprise resource planning) modules is curated by the Belgium-based Odoo S.A. Additionally, customized programming, support, and other services are provided by an active global community and a network of 500 official partners. The main Odoo components are the OpenObject framework, about 30 core modules (also called official modules) and more than 3000 community modules

Odoo has been used as a component of university courses. A study on experiential learning suggested that OpenERP provides a suitable alternative to proprietary systems to supplement teaching.

Several books have been written about Odoo, some covering specific areas such as accounting or development

Odoo has received awards including Trends Gazelle and BOSSIE Awards three years in a row.

It uses Python scripting and PostgreSQL as it's database. Its community edition is supplemented with an Enterprise edition @ USD 240/- per user per year and a commercially supported online edition.The development repository is on GitHub.

In 2013, the not-for-profit Odoo Community Association was formed to ensure the ongoing promotion and maintenance of the Odoo community versions and modules to supplement the work of Odoo S.A. This organisation has over 150 members who are a mix of individuals and organisations.

Read Getting started with odoo-8 online: https://riptutorial.com/odoo-8/topic/2151/getting-started-with-odoo-8

Chapter 2: Add CSS and Javascript files to Odoo module

Syntax

- Note about XML syntax: As the record is made inside of XML file, you can not leave any tag unclosed as you could in a plain HTML, like: <link rel='stylesheet' href="..." **>**, Close the link tag instead, like:
    - <link rel='stylesheet' href="..." **/>**

Parameters

Possible values of *inherit_id* parameter	meaning
web.assets_backend	Used in internal pages only, NOT included in a public website.
website.assets_frontend	Used in a public website only (via "*website*" module).
web.assets_common	Used in both, public website and internal pages.

Remarks

If you are not sure about which option is suitable for you, then try the first option (backend) as it is used in most cases and nearly in all cases if you have not installed the "website" module. Odoo differentiates between "backend" and "frontend" assets because the public website provided by the "website" module uses different styling and JS code than internal pages meant to be used for ERP tasks, i.e. "frontend" is associated with the public website and "backend" is associated with internal pages for ERP (meaning of "frontend" and "backend" are Odoo specific here, but they are both "frontend" in more general sense).

You can not only choose and use one of the options, but also use any combination of them (two of them or all of them) in the same module. Factor a backend, a frontend and a common JS/CSS code into separated files to better adhere to DRY and have suitable code in the public website and in the internal pages.

Do not forget to add "web" (when using *option 1*) or "website" (when using *option 2*) to the dependency list in the `__openerp__.py` manifest.

Examples

Store CSS and JS files correctly in Odoo module

CSS and JS files should be reside under 'static' directory in the root directory of module (the rest of subdirectory tree under 'static' is an optional convention):

- static/src/css/**your_file.css**
- static/src/js/**your_file.js**

Then add links to these files unsing one of the 3 ways listed in the following examples.

Option 1: [BACKEND] Add CSS and Javascript files to use in internal pages

Odoo v8.0 way is to add corresponding record in the XML file:

- Add XML file to the manifest (i.e. `__openerp__.py` file.):

  ...
  'data' : ['your_file.xml'],
  ...

- Then add following record in `'your_file.xml'`:

```
    <openerp>
        <data>
            <template id="assets_backend" name="your_module_name assets"
inherit_id="web.assets_backend">
                <xpath expr="." position="inside">
                    <link rel='stylesheet' href="/your_module_name/static/src/css/
your_file.css"/>
                    <script type="text/javascript" src="/your_module_name/static/src/js/
your_file.js"></script>
                </xpath>
            </template>
        ....
        ....
        </data>
     </openerp>
```

Option 2: [FRONTEND] Add CSS and Javascript files to use in a public website

Note: you should use this way if you've installed a "website" module and you have a public website available.

- Add following record in `'your_file.xml'`:

```
    <openerp>
        <data>

            <template id="assets_frontend" name="your_module_name assets"
inherit_id="website.assets_frontend">
                <xpath expr="link[last()]" position="after">
```

```
                    <link rel='stylesheet' href="/your_module_name/static/src/css/
your_file.css"/>
                </xpath>
                <xpath expr="script[last()]" position="after">
                    <script type="text/javascript" src="/your_module_name/static/src/js/
your_file.js"></script>
                </xpath>
            </template>

        </data>
    </openerp>
```

Option 3: [COMMON] Add CSS and Javascript files to use in all pages (backend & frontend)

- Add following record in `'your_file.xml'`:

```
    <openerp>
        <data>

            <template id="assets_common" name="your_module_name assets"
inherit_id="web.assets_common">
                <xpath expr="." position="inside">
                    <link rel='stylesheet' href="/your_module_name/static/src/css/
your_file.css"/>
                    <script type="text/javascript" src="/your_module_name/static/src/js/
your_file.js"></script>
                </xpath>
            </template>

        </data>
    </openerp>
```

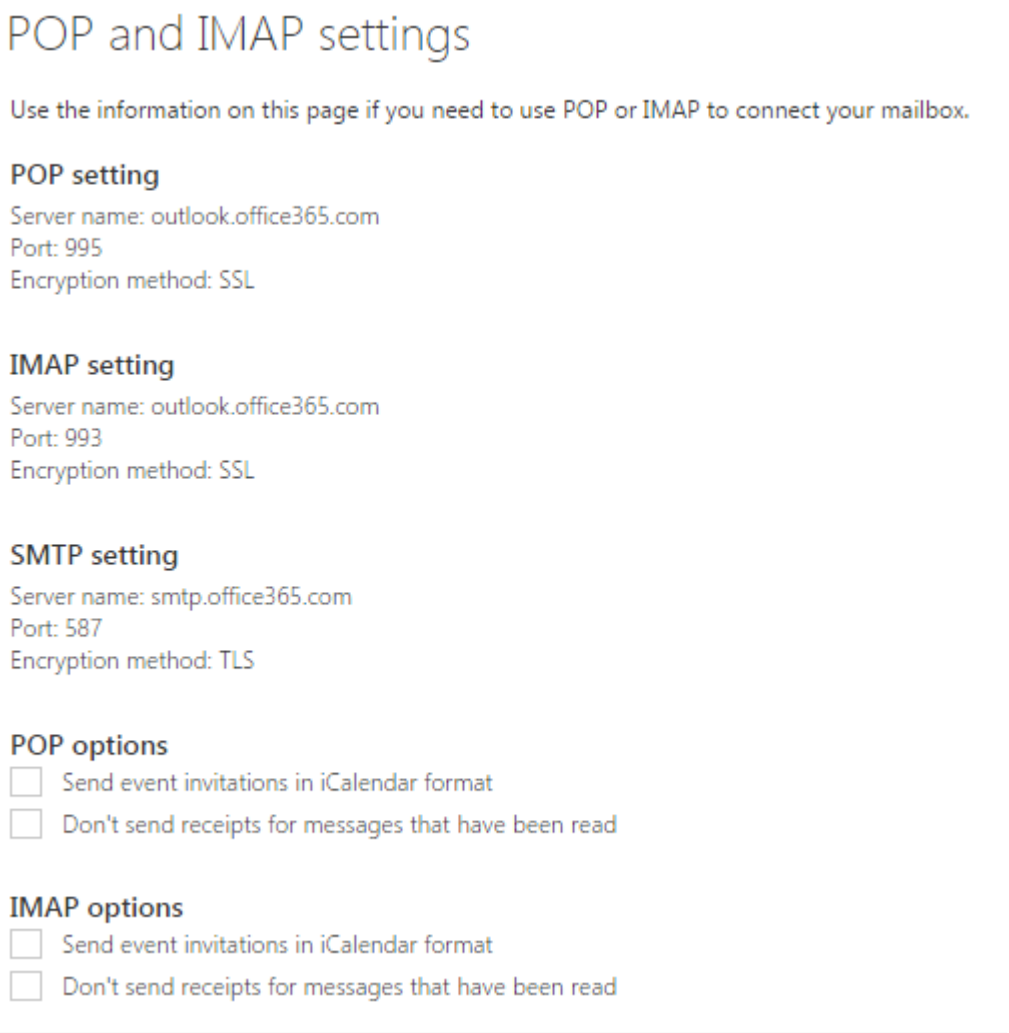
Read Add CSS and Javascript files to Odoo module online: https://riptutorial.com/odoo-8/topic/3401/add-css-and-javascript-files-to-odoo-module

Chapter 3: Configure Email - Office 365 in Odoo

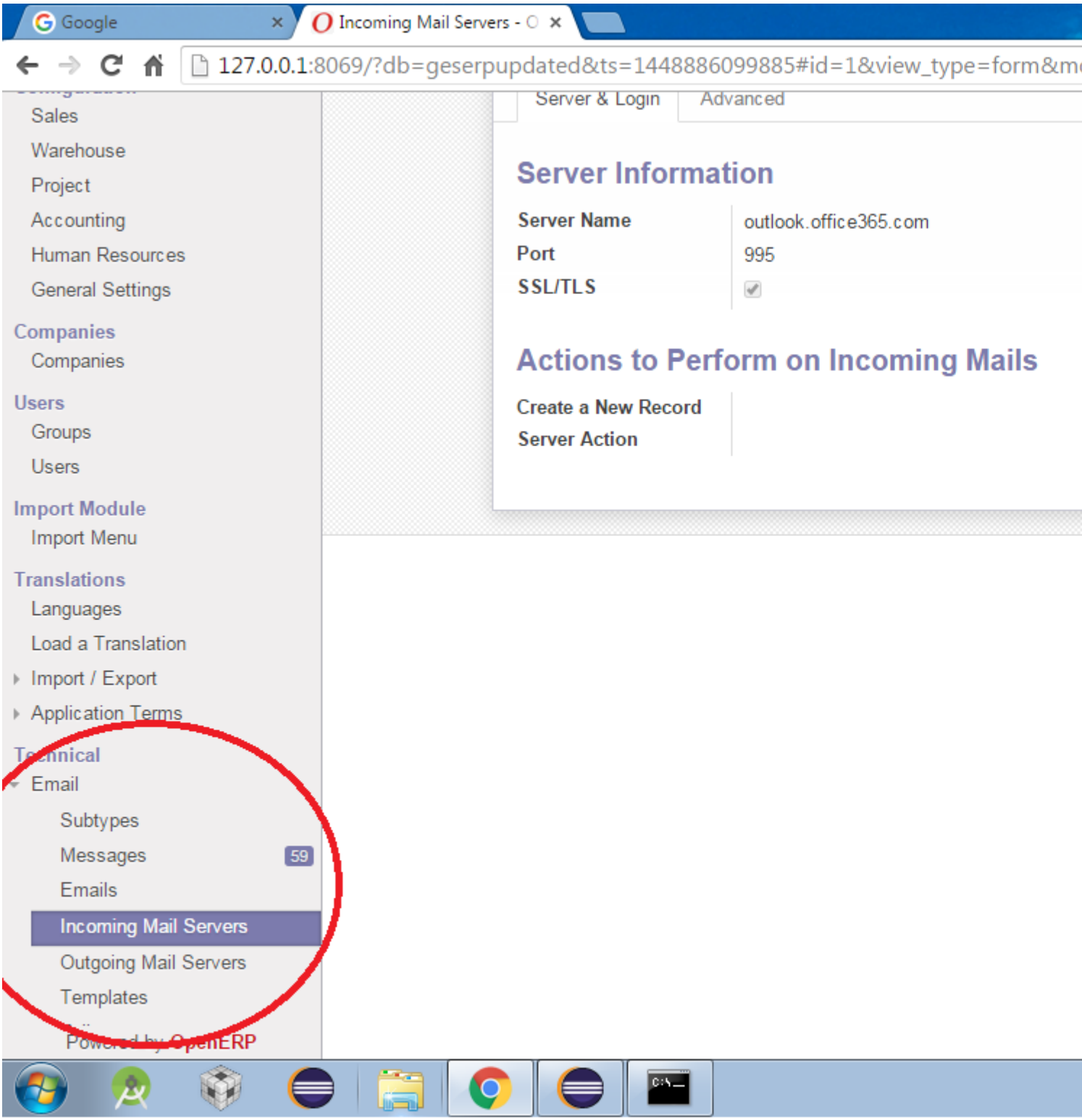
Examples

Configure E-Mail

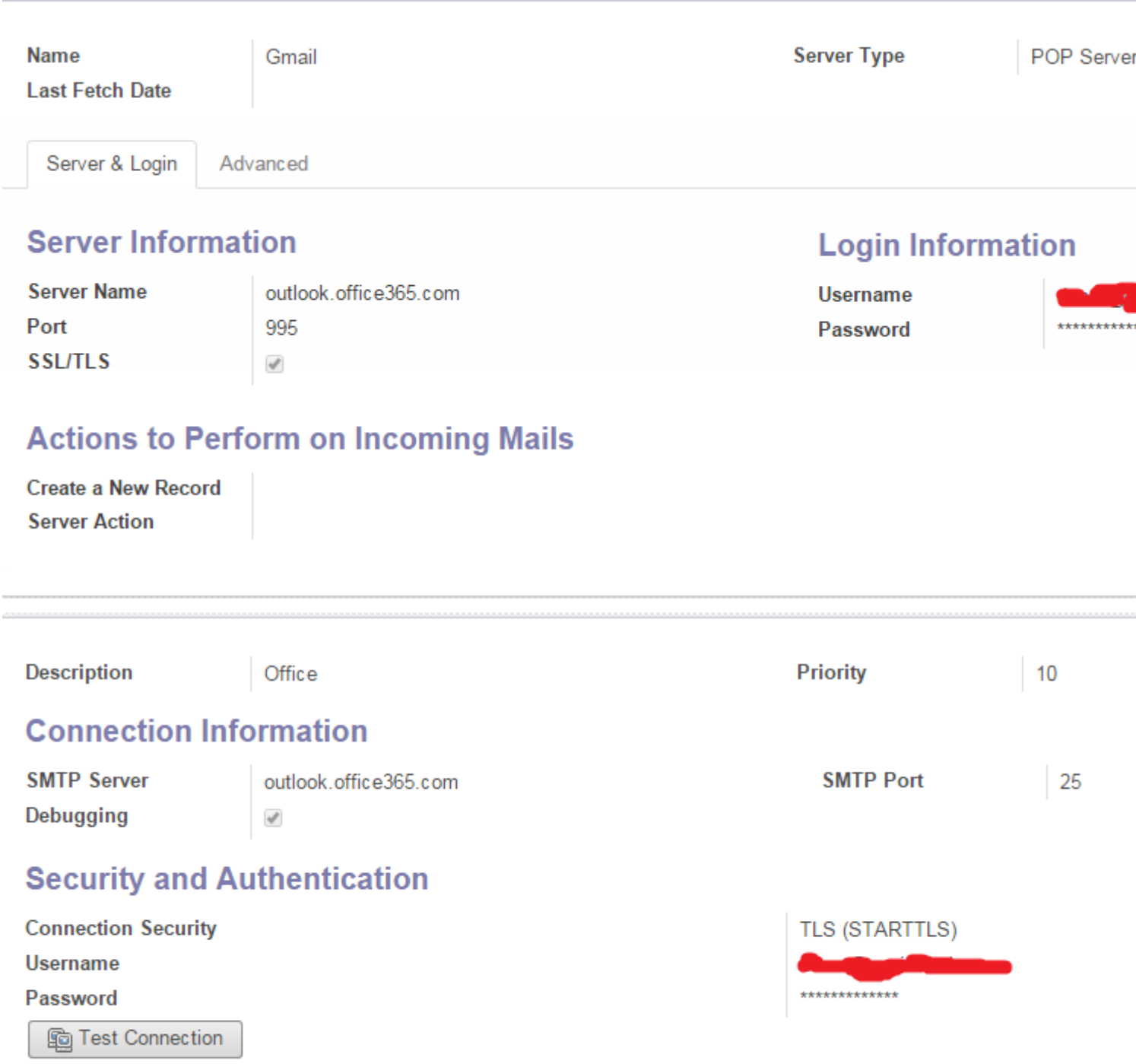
- Initially check your E-Mail Settings



- **In Odoo go to Settings --> Email .**



- **Enter the field values in "Incoming Mail Servers" & "Outgoing Mail Servers" Options.**



Read Configure Email - Office 365 in Odoo online: https://riptutorial.com/odoo-8/topic/6648/configure-email---office-365-in-odoo

Chapter 4: Create Automated Functions For Model

Introduction

We often need to run some code automatically during module install. This have many reasons for example configuring `Sale` module settings to meet our project requirements.

In this topic you will learn how to make automated function run on module install.

Examples

First of all you need to create xml file for make function call

```
<?xml version="1.0"?>
<openerp>
    <data noupdate="1">
        <function model="*model_name*" name="_configure_sales"/>
    </data>
</openerp>
```

This simple xml file is calls `_configure_sales` function from *model_name* model.

NOTE: this xml file should be on the top of `data` array, because Odoo is processiong xml files from top to bottom.

Corresponding Python file

```
class *model_name*(models.Model):
    _name = *model_name*

    @api.model
    def _configure_sales(self):
        # Do the configuration here
```

Every time when module will be installed this function will run.

Note: If you remove `noupdate` from xml, function will run on upgrading as well.

Read Create Automated Functions For Model online: https://riptutorial.com/odoo-8/topic/10633/create-automated-functions-for-model

Chapter 5: Custom widgets for fields

Remarks

- make sure you properly add javascript file to your module
- do not forget to add 'web' as dependency in __openerp__.py:

```
'depends': ['web',....]
```

Examples

Custom widget for numeric fields to use in TreeView

The below example widget demonstrates how to format individual cells of a TreeView column conditionally, depending on value of the field in the particular cell. If value of field is negative, then it'll be displayed in red color and minus symbol will be hidden, otherwise it'll be displayed in normal color .

A widget should be written in JavaScript, lets use `custom_widget_name` as a name for a new widget, and `your_module_name` is a technical name of your module (same as your module's root directory name)

Uunder static/src/js/ folder in your module add javascript file (say static/src/js/*custom_widget*.js) with a custom widget in it:

```
 openerp.your_module_name = function (instance) {

    instance.web.list.columns.add('field.custom_widget_name',
'instance.your_module_name.custom_widget_name');

    instance.your_module_name.custom_widget_name = instance.web.list.Column.extend({
        _format: function (row_data, options) {
            res = this._super.apply(this, arguments);
            var amount = parseFloat(res);
            if (amount < 0){
                return "<font color='#ff0000'>"+(-amount)+"</font>";
            }
            return res
        }
    });
    //
    //here you can add more widgets if you need, as above...
    //
};
```

the above example widget can be used in a list view for field of type float and it applies custom rules as follows:

- Negative numbers:

    - Are shown in red.
    - Minus symbol (a '-' character) is "hidden".
- For positive numbers default layout is used.

This example widget can be applied to a field in a tree view of Odoo. You can use widget like this for a column you need to apply the custom rules to:

```
. . .
<tree >
    . . .
    <field name="some_field" widget="my_widget" />
    . . .
</tree>
. . .
```

Read Custom widgets for fields online: https://riptutorial.com/odoo-8/topic/6198/custom-widgets-for-fields

Chapter 6: Fields used in Odoo 8

Introduction

This is section where you can find the details about the fields that is being used in Odoo 8

Parameters

Parameters	Description
string="Name"	Optional label of the field
compute="_compute_name_custom"	Transform the fields into computed fields
store=True	If computed it will store the result
select=True	Force index on field
readonly=True	Field will be readonly in views
inverse="_write_name"	On update trigger
required=True	Mandatory field
translate=True	Translation enable
help='blabla'	Help tooltip text
comodel_name="model.name"	Name of the related model
inverse_name="field_name"	relational column of the opposite model
relation='many2many_table_name'	relational table name for many2many
columns1='left_column_name'	relational table left column name
column2='right_column_name'	relational table right column name

Remarks

Odoo and ORM: Odoo uses ORM(Object Relational Mapping) technique to interact with database. ORM will help to create a virtual object database that can be used within from the Python. In ORM technique each model is represented by a class that subclasses Models.model.

Models.model is the main super class for regular database persisted Odoo models. Odoo models are created by inheriting from this class.

Example:

```
class Employee(Models.model):
    _name = 'module.employee'

    #Rest of the code goes here
```

Here _name is a structural attribute, which tells the system about the name of the database table to be created.

Each model has a number of class variables, each of which represents a database field in the model. Each field is represented by an instance of a openerp.fields.Field class. Fields in Odoo are listed below..

1 Boolean Field

```
ex: flag = fields.Boolean()
```

2 Char Field

```
ex: flag = fields.Char()
```

3 Text

```
ex: flag = fields.Text()
```

4 Html

```
ex: flag = fields.Html()
```

5 Integer

```
ex: flag = fields.Integer()
```

6 Float

```
ex: flag = fields.Float()
```

7 Date

```
ex: flag = fields.Date()
```

8 Datetime

```
ex: flag = fields.Datetime()
```

9 Selection

```
ex: flag = fields.Selection()
```

10 Many2one

```
ex: flag = fields.Many2one()
```

11 One2many

```
ex: flag = fields.One2many()
```

12 Many2many

```
ex: flag = fields.Many2many()
```

Examples

Examples fields of Odoo 8

Odoo uses ORM(Object Relational Mapping) technique to interact with database. ORM will help to create a virtual object database that can be used within from the Python. In ORM technique each model is represented by a class that sub-classes Models.model. Models.model is the main super class for regular database persisted Odoo models. Odoo models are created by inheriting from this class

```
name = fields.Char(string='New Value')

flag = fields.Boolean(string='Flag',default=False)

amount = fields.Float(string='Amount',digits=(32, 32))

code = fields.Selection(string='Code',selection=[('a', 'A'),('b','B')])

customer = fields.Many2one(comodel_name='res.users')

sale_order_line = fields.One2many(comodel_name='res.users', inverse_name='rel_id')

tags = fields.Many2many(comodel_name='res.users',
                        relation='table_name',
                        column1='col_name',
                        column2='other_col_name')
```

Read Fields used in Odoo 8 online: https://riptutorial.com/odoo-8/topic/8152/fields-used-in-odoo-8

Chapter 7: How to activate OpenERP Developer Mode

Remarks

Developer Mode

Odoo developer mode allows you to make substantial modifications to the Odoo database such as adding fields to your documents and views. You change the default views of your actions and can even create dynamic forms based on other fields within your models.

Advantage

While Odoo is a powerful application framework the development cycle can be brutal to test changes to your application. By utilizing the developer mode you can test expressions and solve many functional problems without having to restart the server over and over to test simple changes.

Additionally the Odoo developer tool is great for looking at the architecture of forms and views to see how fields are tied to modules, their domains, contexts and other attributes. In this video we explore exactly how we put these tools to use in modifying and creating Odoo applications.

Limitations

While it can be very tempting to use developer mode to make a great deal of changes to your application there are some drawbacks. Depending on what you modify and change you can lose these changes with future module updates or when you install additional applications into Odoo. This is particularly true for changes to views.

> To activate developer mode you just simply write down
>
> for version v7
>
> &debug=
>
> before # sign you just add it.
>
> http://localhost:8069/?db=test_db&debug=#
>
> for version > v7
>
> http://localhost:8069/web?debug=

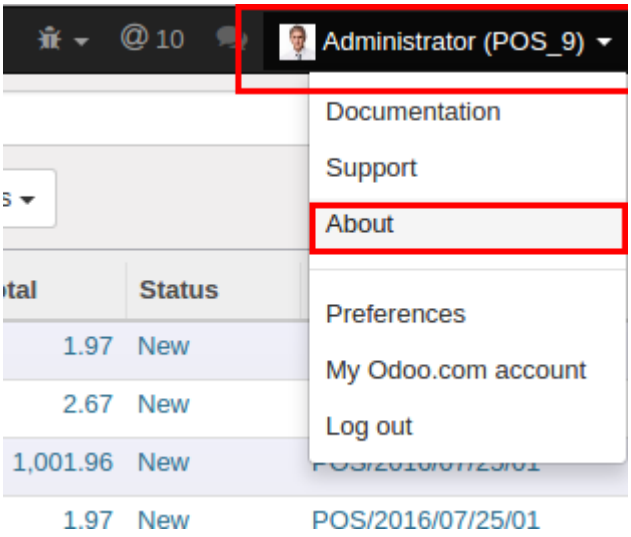
You may not see **About Odoo** menu because there might be **odoo debranding module** installed.

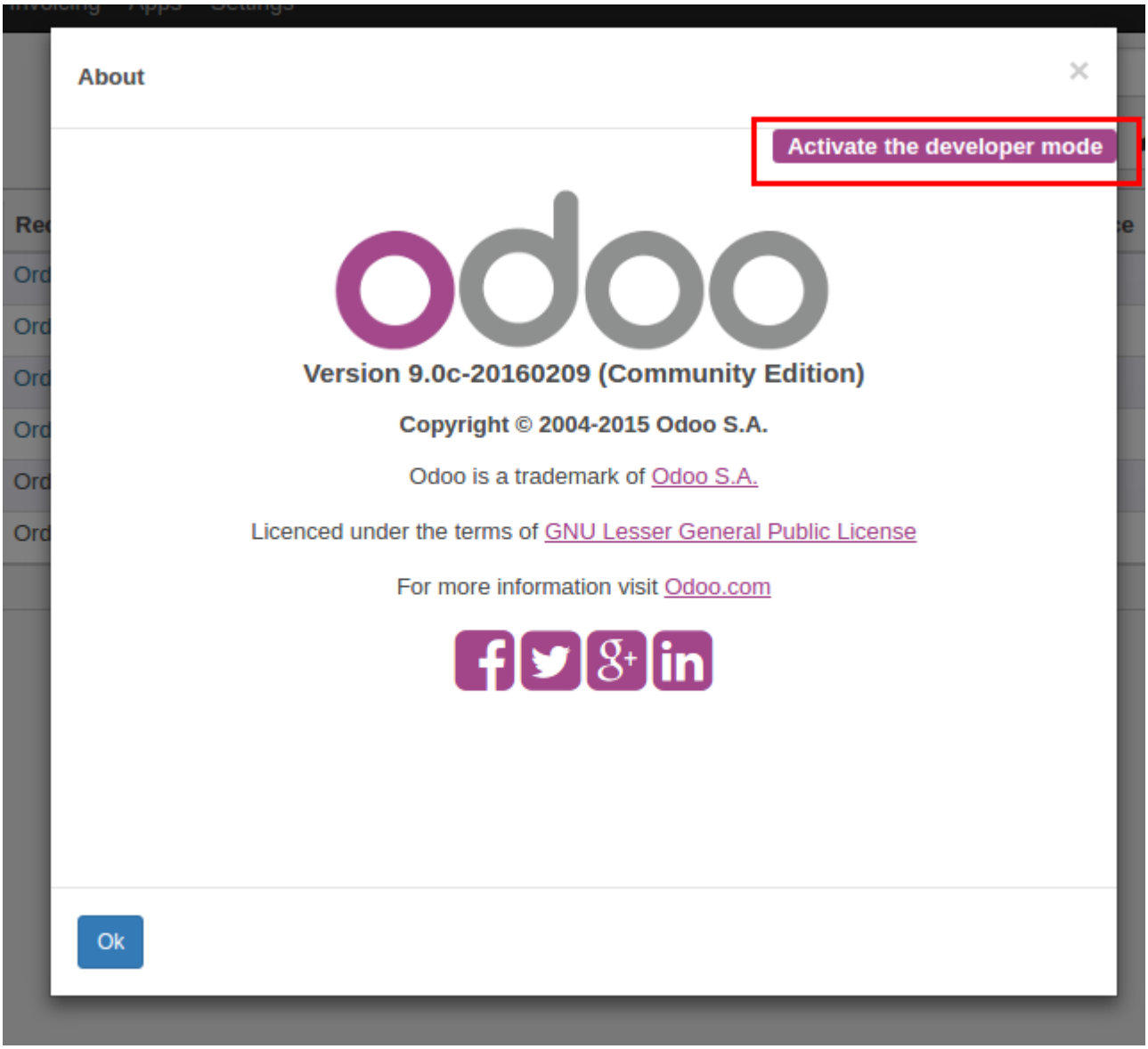
Examples

Activate developer mode

To activate the developer mode:

1. Log in to the ODOO front end
2. Click on the User Name drop down at the top-right side
3. Select 'About'
4. Click on 'Activate developer mode' from the pop-up window.





Activating developer mode in Odoo 8

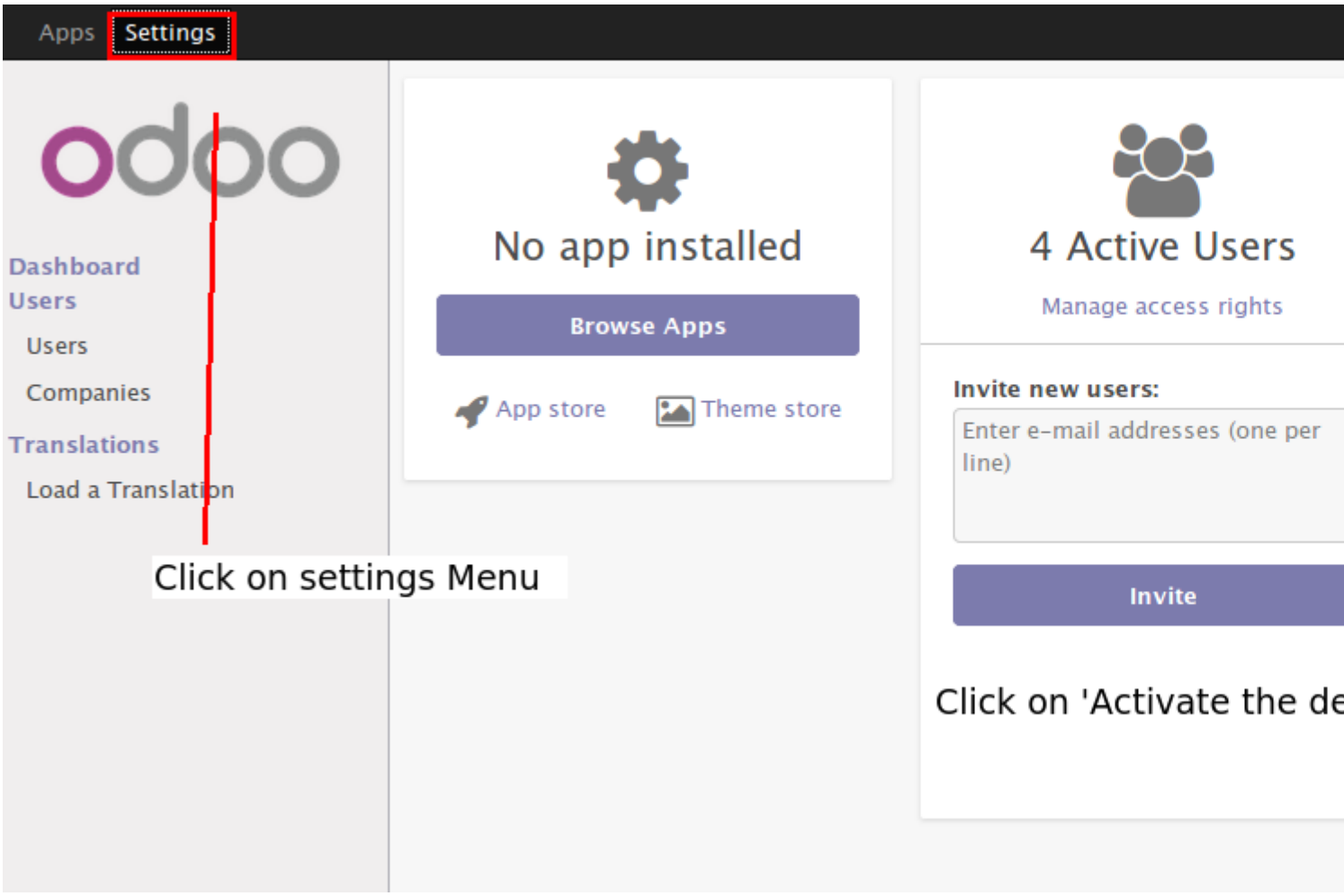
When you logged into Odoo application you will find an option to see who is the current logged in person in the top right corner. This user information have a dropdown button. Click on the dropdown, then you will find a list. In that list select about Odoo.com option. Clicking on that will open a **About** popup window. In that window, over the top right corner you will find an option like **Acivate developer mode**. Clicking on that link will reload the webpage.

After reloading it will be in Developer mode. Then the link will change to something like this *http://localhost:8069/web?debug=#id=23&view_type=form&model=res.partner*

Activate developer mode in Odoo 10

Activate Developer Mode:

1. Login to odoo application.
2. After login user may see several odoo menu's. Click on setting menu.



1. Click on 'Activate the developer mode' which is located at right-bottom corner of settings page.
2. Developer mode now activated.

Read How to activate OpenERP Developer Mode online: https://riptutorial.com/odoo-8/topic/3311/how-to-activate-openerp-developer-mode

Chapter 8: RPC using Odoo v8 API (Call Python function from JavaScript)

Remarks

If you are considering to add new methods in Python to use them in RPC from JavaScript, then consider the following options of method decorators: if you've to deal with ids/recordsets then for python method definition choose decorator:

- @api.multi - to get *recordset* in your method
- @api.one - to get *browse_records* one by one in your method in above examples @api.multi is used, but @api.one also may be used to deal with ids, depending on requirements (However, it's strongly recommended to use @api.multi instead of @api.one for performance reasons).

Or if it's simple function that does not have to deal with records/ids then for python method choose decorator:

- @api.model - Allows to be polite with old style API.
- @api.multi - Again, you can use it here as well, just pass `[ ]` (empty array) as first argument in javascript...

References: Odoo RPC documentation, Odoo 8 API method decorators

Examples

An example Odoo model to call methods from

```
class my_model(models.Model):
    _name = "my.model"

    name = fields.Char('Name')

    @api.multi
    def foo_manipulate_records_1(self):
        """ function returns list of tuples (id,name) """
        return [(i.id,i.name) for i in self]

    @api.multi
    def foo_manipulate_records_2(self, arg1, arg2)
        #here you can take advantage of "self" recordset and same time use aditional arguments
"arg1", "arg2"
        pass

    @api.model
    def bar_no_deal_with_ids(self, arg1, arg2):
```

```
        """ concatenate arg1 and arg2 """
        return unicode(arg1) + unicode(arg2)
```

Odoo RPC examples

Examples below demonstrate how to call Python function from JavaScript in Odoo 8. In the examples we call methods of *my_model* described early on this page.
We assume that in the following examples "list_of_ids" variable contains list(array) of ids of existing records of "my.model" model.

- Call of method **foo_manipulate_records_1** decorated with **@api.multi**:

```
new instance.web.Model("my.model")
    .call( "foo_manipulate_records_1", [list_of_ids])
        .then(function (result) {
            // do something with result
    });
```

- Call of method **foo_manipulate_records_2** decorated with **@api.multi**:

```
new instance.web.Model("my.model")
    .call( "foo_manipulate_records_2", [list_of_ids, arg1, arg2])
        .then(function (result) {
            // do something with result
    });
```

- Call of method **bar_no_deal_with_ids** decorated with **@api.model**:

```
new instance.web.Model("my.model")
    .call( "bar_no_deal_with_ids", [arg1, arg2])
        .then(function (result) {
            // do something with result
    });
```

Also if it has some sense depending on implementation, then you can call function decorated with @api.multi even if you have not to deal with ids (just pass empty array in place of ids, as first element of argument list):

```
new instance.web.Model("my.model")
    .call( "foo_manipulate_records_2", [[], arg1, arg2])
        .then(function (result) {
            // do something with result
    });
```

this way may be useful in some cases, as undecorated function in v8.0 api is considered as @api.multi (as @api.multi is a default decorator)

Cxcept of two parameters to RPC call that are used in the above examples (the function name and argument list), you can use **third parameter** - a **dictionary of keyword arguments**. It's highly recommended to turn around a context (in some cases it might be even necessary), as it may change behavior of remote procedure (localization, etc.). See below the example with context argument in RPC call (same may be applied to all examples above)

```
var self = this;
new instance.web.Model("my.model")
    .call("foo_manipulate_records_2", [[], arg1, arg2], {'context':self.session.user_context})
        .then(function (result) {
            // do something with result
    });
```

Of course you can use custom context as well, if necessary, instead of turning around the existing one as in this example.

Read RPC using Odoo v8 API (Call Python function from JavaScript) online: https://riptutorial.com/odoo-8/topic/6613/rpc-using-odoo-v8-api--call-python-function-from-javascript-

Chapter 9: What are the ORM Methods and details?

Remarks

Create method: Create new record with specified value. Takes a number of field values, and returns a recordset containing the record created

```
def create(self,vals):
    return super(class_name, self).create(vals)
```

Write Method: Update records with given ids with the given field values.Takes a number of field values, writes them to all the records in its recordset. Does not return anything

```
def write(self,vals):
    return super(class_name, self).write(vals)
```

Search method: Search for records based on a search domain.Takes a search domain, returns a recordset of matching records. Can return a subset of matching records (offset and limit parameters) and be ordered (order parameter)

```
self.search([('customer','=',True)])
self.env['res.partner'].search(['partner','=',True])
```

Browse method: Fetch records as objects allowing to use dot notation to browse fields and relations.Takes a database id or a list of ids and returns a recordset, useful when record ids are obtained from outside Odoo (e.g. round-trip through external system) or when calling methods in the old API.

```
self.browse([7,8,9])
self.env['res.partner'].browse([7,8,9])
```

Exists methods: Returns a new recordset containing only the records which exist in the database. Can be used to check whether a record (e.g. obtained externally) still exists.

```
records = records.exists()
```

ref method: Environment method returning the record matching a provided external id

```
self.env.ref('base.group_public')
```

ensure_one method: checks that the recordset is a singleton (only contains a single record), raises an error otherwise

```
records.ensure_one()
```

Examples

Different types of ORM Methods

1. create()
2. write()
3. search()
4. browse()
5. exists()
6. ref()
7. ensure_one()

Read What are the ORM Methods and details? online: https://riptutorial.com/odoo-8/topic/6150/what-are-the-orm-methods-and-details-

Credits

S. No	Chapters	Contributors
1	Getting started with odoo-8	4444, Community, danidee, T.V.
2	Add CSS and Javascript files to Odoo module	George Daramouskas, T.V.
3	Configure Email - Office 365 in Odoo	Don Chakkappan
4	Create Automated Functions For Model	Dachi Darchiashvili
5	Custom widgets for fields	T.V.
6	Fields used in Odoo 8	AKHIL MATHEW
7	How to activate OpenERP Developer Mode	AKHIL MATHEW, Emipro Technologies Pvt. Ltd., Gopakumar N G, Mehedi Hasan
8	RPC using Odoo v8 API (Call Python function from JavaScript)	T.V.
9	What are the ORM Methods and details?	AKHIL MATHEW, Mani