Development Concepts

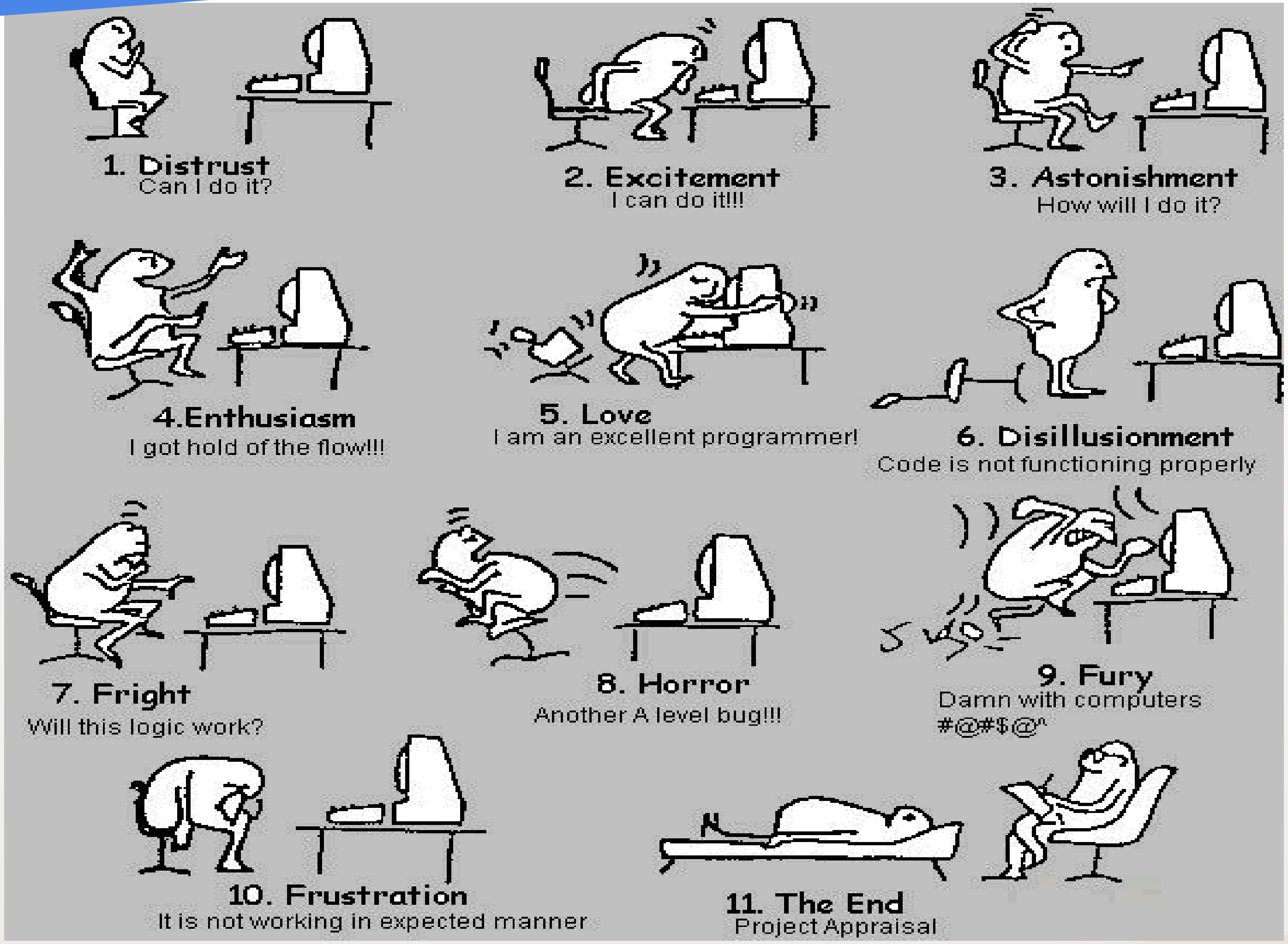
Odoo Framework

Complex is better than complicated

OBJECTIVES

- Understand developers

UNDERSTAND DEVELOPERS



OBJECTIVES

- Understand the development concepts and architecture

SUMMARY

- Odoo Architecture
- High Performance
- Sources Organisation
- Module Structure
- Models
- Fields
- Recordset
- API
- Methods
- Search Domain
- Exceptions
- Cache Management
- Developer Mode
- Data Import
- Menus & Actions
- Views
- Structural Components
- Semantic Components
- Messaging Features
- Reports (Qweb)
- Reports (SQL view)
- Workflows
- Security
- Translations

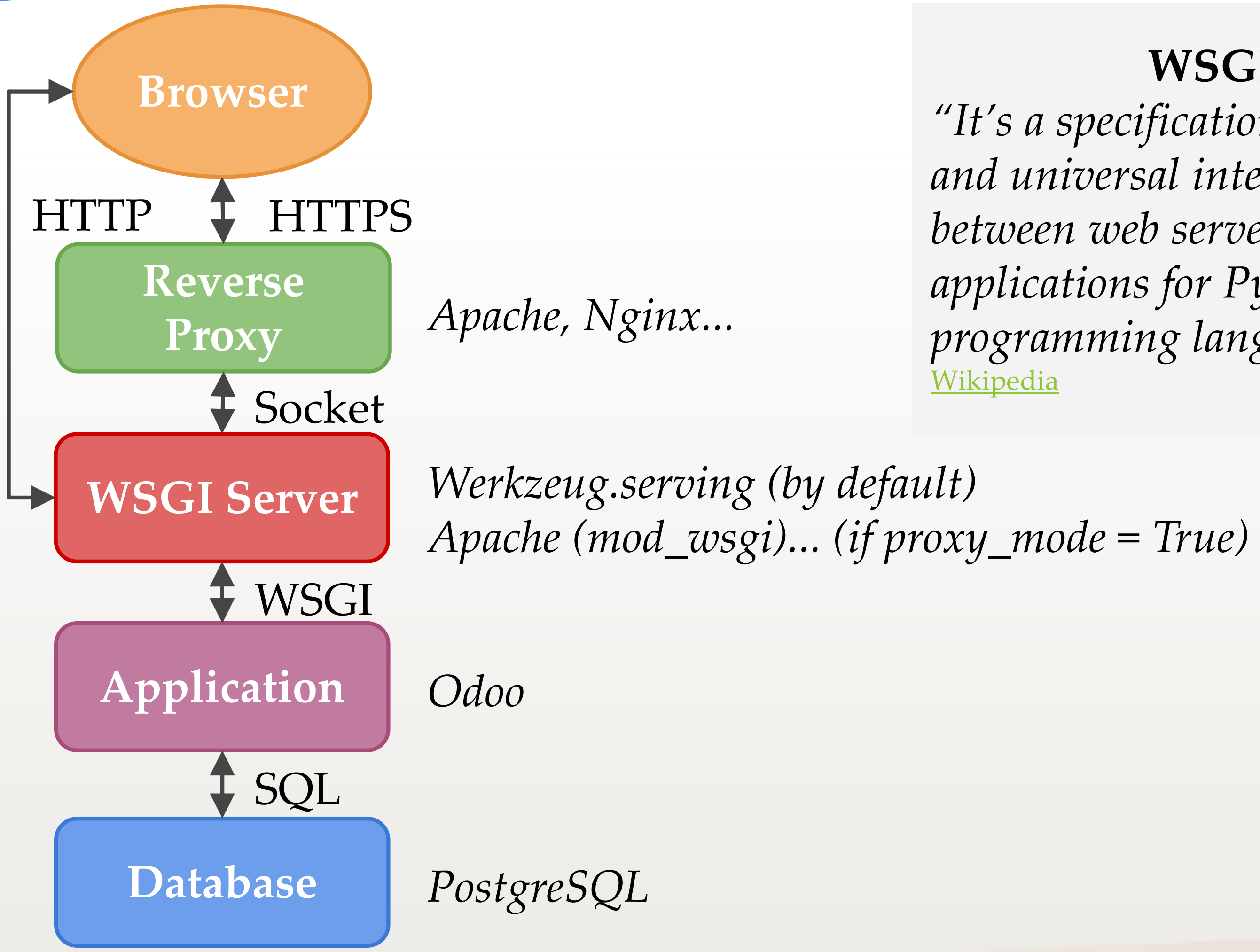
SUMMARY

- Tests (Unit Tests)
- Tests (JS Tours)
- Web Services (XML/RPC)
- Web Services (JSON/RPC)
- Web Controllers
- Javascript TODO
- Website Themes & Snippets TODO
- Odoo Studio
- Website Editor
- Multi-Company
- Gamification
- Keyboard Shortcuts
- Barcordes
- Odoo Tools
- Odoo Date Utils
- Asynchronous
- Coding Guidelines
- Best Practices
- Perfs Analysis

REQUIREMENTS

- Enthusiasm

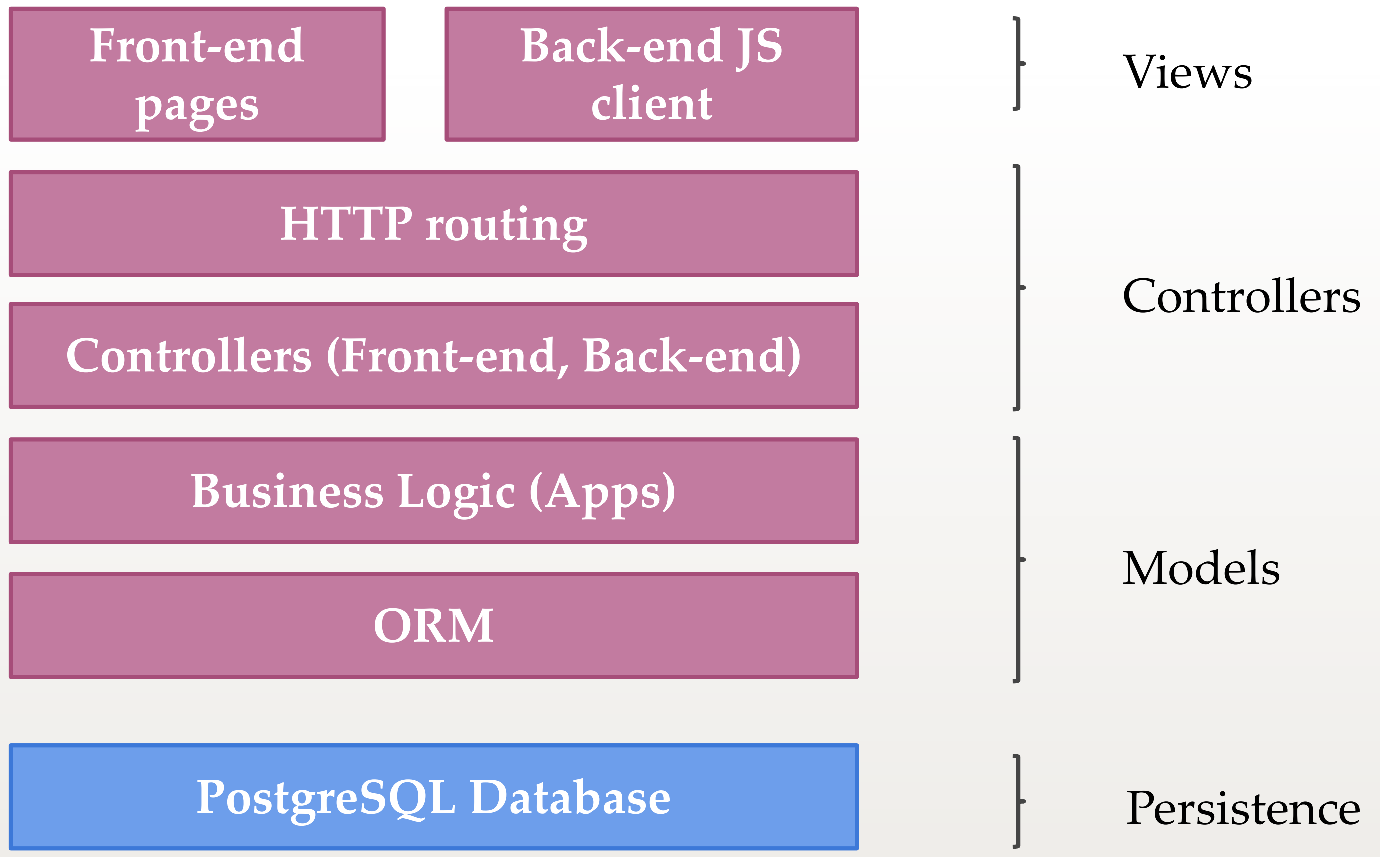
ARCHITECTURE



WSGI

"It's a specification for simple and universal interface between web servers and web applications for Python programming language", source: Wikipedia

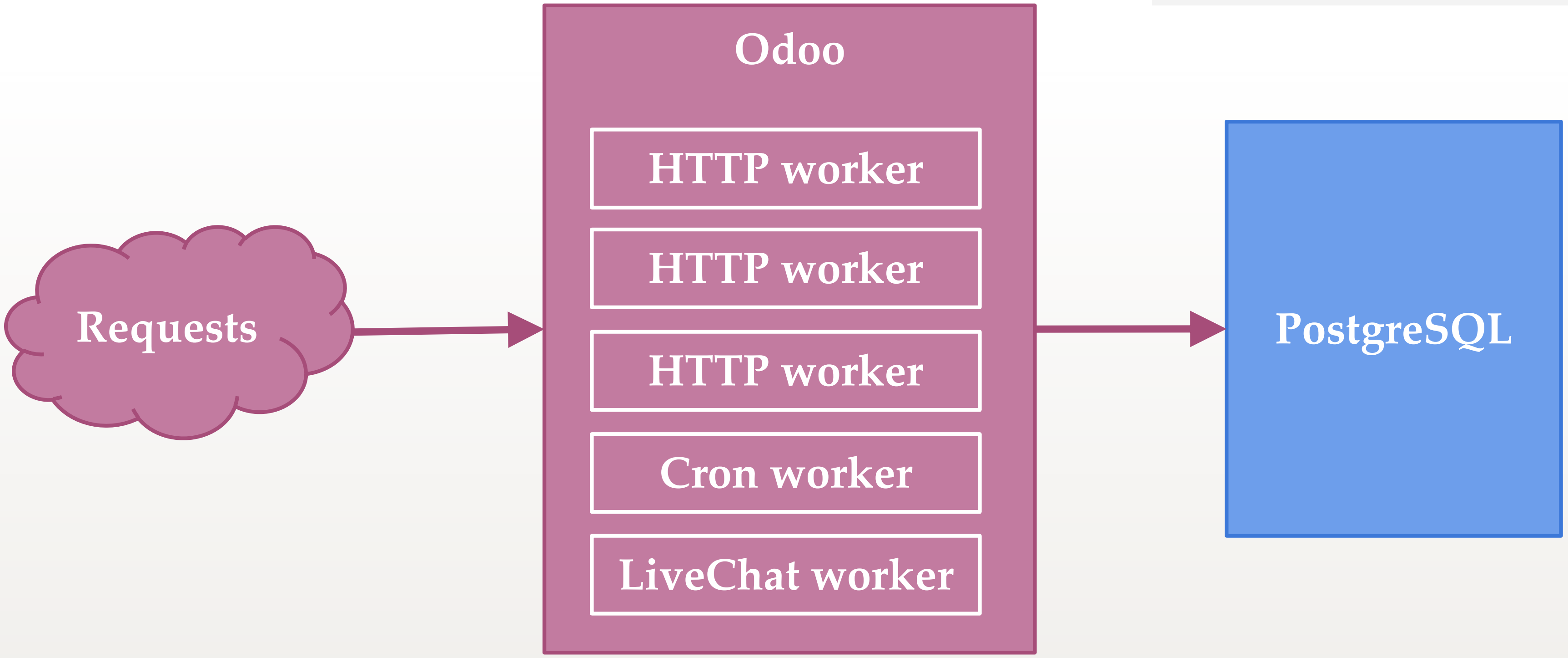
ARCHITECTURE



HIGH PERFORMANCE

Multi Workers

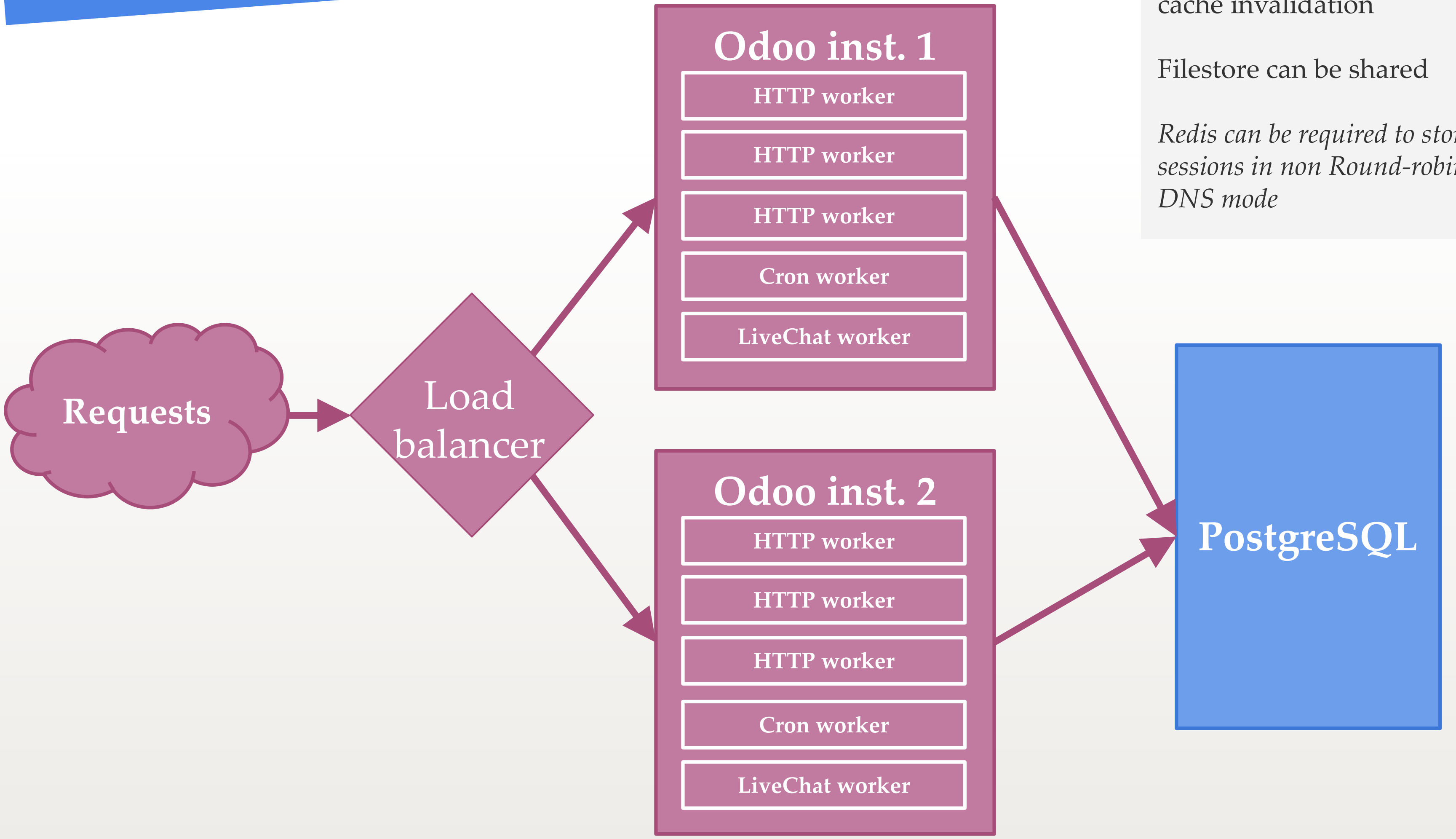
To support more users
Not to accelerate business processes



Rule of thumb : (#CPU * 2) + 1

See documentation

HIGH PERFORMANCE



Load balancing

Multi-workers required for cache invalidation

Filestore can be shared

Redis can be required to store sessions in non Round-robin DNS mode

SOURCES ORGANIZATION

A Odoo project repository respects this organization

Notes:

- *server is a nightly build unzipped archive*
- *extra-addons are Odoo modules developped by community*
- *upgrades are specific Python modules to auto-upgrade database*

server/

enterprise-addons/

***project*-addons/**

etech-addons/

extra-addons/

upgrades/

MODULE

One **module** = One **feature**

One business app = One module with *application: True* in manifest

```
my_module/
     controllers/
          __init__.py
          main.py
     data/
          object_name_data.xml
          object_name_data.yml
          object.name.csv
          object_name_demo.xml
     i18n/
          fr.po
          my_module.pot
     models/
          __init__.py
          object_name.py
     report/
          object_name_report.xml
     security/
          ir.model.access.csv
          my_module_security.xml
          ir_rule.xml
```

```
     static/
          description/
               icon.png
          lib/
          scr/
               css/
               js/
               xml/
     test/
          my_test.yml # deprecated
     tests/
          __init__.py
          my_test.py
     views/
          object_name_view.xml
          menus.xml
     wizard/
          __init__.py
          my_wizard.py
          my_wizard_view.xml
     workflow/
          object_name_workflow.xml
     __init__.py
     __manifest__.py
```

MANIFEST

Odoo manifest = __manifest__.py

(previously __openerp__.py or __terp__.py)

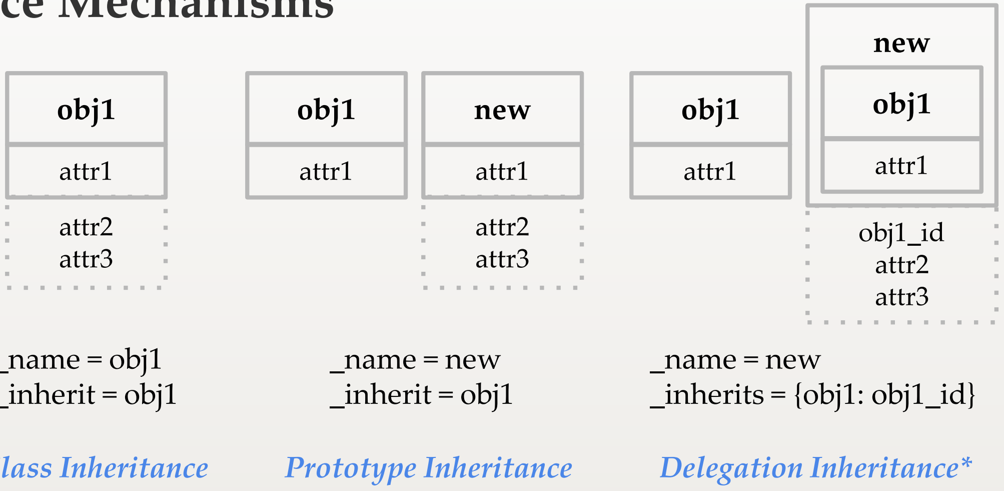
```
# -*- coding: utf-8 -*-

{
    'name': 'My Module',
    'category': 'Accounting',
    'summary': 'My Module Summary',
    'website': '',
    'version': '1.0',
    'license': 'AGPL-3',
    'description': """
My Module Description
=====================


        """,
    'author': 'Myself',
    'depends': ['base'],
    'data': [
        'data/object_name.xml',
        'report/object_name_report.xml',
        'security/ir.model.access.csv',
        'security/my_module.xml',
        'views/object_name_view.xml',
        'wizard/my_wizard_view.xml',
        'workflow/object_name_workflow.xml',

    ],
    'demo': [
        'data/object_name_demo.xml',
    ],
    'test': [
        'test/my_test.yml',
    ],
    'qweb': [
        'static/src/xml/...xml'
    ],
    'application': False,
    'auto_install': False,
    'installable': True,
    'external_dependencies': {
        'bin': [],
        'python': [],
    },
}
```

MODELS

Types of object models

- **Model**: database persisted
- **TransientModel**: temporarily db persisted
- **AbstractModel**: db non-persisted

Inheritance Mechanisms



** When using delegation inheritance, methods are not inherited, only fields*

MODELS

```
# -*- coding: utf-8 -*-
# License

from odoo import models

class MyModel(models.Model)
    _name = ‘my.model’
    _description = ‘My Model’
    _auto = True  # Auto-create table
    _table = ‘my_model’  # Table name, by default is equals to _name.replace(‘.’, ‘_’)
    _sql = ‘’  # SQL code to create table/view
    _inherit = []
    _inherits = {}
    _columns = {fields_name: fields_instance}  # Deprecated::7.0
    _defaults = {fields_name: default_value}  # Deprecated::7.0
    _order = ‘id’
    _log_access = True  # Create 4 columns: create_uid, create_date, write_uid, write_date
    _rec_name = ‘name’  # Used in name_get() to display record name
    _constraints = []  # Deprecated::7.0
    _sql_constraints = [(constraint_name, constraint_sql_code, constraint_warning)]
    _parent_store = False  # Use a modified preorder tree traversal
    _parent_name = ’parent_id’
    _parent_order = False
    _date_name = ’date’  # Date field fo default calendar view
    _translate = True  # set to False to disable translations export for this model
```

FIELDS

Simple Fields

- Char
- Text
- Html
- Float
- Monetary
- Integer
- Boolean
- Binary
- Datetime
- Date
- Selection
- *Serialized*

Relational Fields

- Many2one
- One2many
- Many2many
- Reference *(polymorphic)*

Fields Attributes

- string: field label
- default: default value, static or callable
- required: required (True/False)
- readonly: readonly in UI (True/False)
- help: bullet help in UI
- index: database index (True/False)
- company_dependent: make value dependent of user main company (True/False)
- copy: copy this field value at record duplication (True/False)
- states: make readonly/invisible in function of record state
- groups: make accessible for some users groups
- related: make reference to the field of a remote object
- related_sudo: execute related as root user (default True)
- compute: name of method to compute field value
- compute_sudo: execute compute as root user
- inverse: name of method to modify computed field value
- search: name of method to search in a computed field not stored
- store: indicate if a computed field is stored in db (True/False)
- group_expand: name of method to expand groups in read_group()
- *context_dependent: new feature in Odoo v11*

FIELDS

Only for string fields (Char, Text, Html)
- size: define max. size - *only for Char*
- translate: make record value translatable

Only for HTML fields
- sanitize: strip the content from potentially insecure tags
- strip_style: remove style element

Only for numeric fields (Float, Integer)
- digits: define decimal precision - *only for Float*
- group_operator: define aggregate operator, *sum by default*

Only for Selection
- selection: [(value1, label1), (value2, label2)...]
- selection_add: idem selection but to add tuples to original selection

*Only for relational fields (*2*)*
- comodel_name: target model _name
- domain: filter accessible records
- context: force context for remote records

Only for relational fields Many2one
- comodel_name: target model _name
- ondelete: ‘set null’ by default, ‘restrict’ to prevent deletion of a referenced record, ‘cascade’ to delete the current record when a referenced record is deleted

Only for relational fields One2many
- comodel_name: target model _name
- inverse_name: name of the inverse ‘Many2one’ field in ‘comodel_name’
- limit: optional limit to use upon read

Only for relational fields Many2many
- comodel_name: target model _name
- relation: name of relation table
- column1: column of table referring to model
- column2: column of table referring to comodel

SPECIAL / RESERVED FIELDS

id	unique system identifier for the object
name	field whose value is used to display the record in lists, etc. if missing, set *_rec_name* to specify another field to use
active	toggle visibility: records with active set to False are hidden by default
sequence	defines order and allows drag&drop reordering if visible in list views
state	lifecycle stages for the object, used by the states attribute
parent_id	defines tree structure on records, and enables child_of operator
parent_left, parent_right	used in conjunction with _parent_store flag on object, allows faster access to tree structures (see also Performance Optimization section)
create_date, create_uid, write_date, write_uid	used to log creator, last updater, date of creation and last update date of the record. disabled if _log_access flag is set to False (created by ORM, do not add them)

FIELDS

```
# -*- coding: utf-8 -*-
# License

from odoo import api, fields, models


class MyModel(models.Model)
    _name = 'my.model'
    _description = 'My Model'

    name = fields.Char(required=True)
    state = fields.Selection([('draft', 'Draft'), ('done', 'Done')],
                             default='draft', readonly=True)
    partner_id = fields.Many2one('res.partner', 'Customer', domain=[('customer', '=', True)],
                                 readonly=True, states={'draft': ['readonly', False]})
    user_ids = fields.Many2many('res.users', 'my_model_res_users_rel', 'my_model_id', 'user_id',
                                'Users')
    line_ids = fields.One2many('my.model', 'my_model_id', 'Lines')
    email = fields.Char(related='partner_id.email')
    users_count = fields.Integer(compute='_get_users_count',compute_sudo=True, store=True)

    @api.one
    @api.depends('user_ids')
    def _get_users_count(self):
        self.users_count = len(self.user_ids)
```

RECORDSETS

A **recordset** is

- an ordered collection of records
- an instance of the model's class

```
partners = self.env['res.partner'].search([])
# res.partner(41, 5, 4, 23, 24)
```

It implements some **sequence and set operations**:

- length *len(self)*
- slicing *[:]*
- union |
- intersection *&*
- difference -
- addition +

```
len(partners)   # 5

partners[0]  # res.partner(41)
partners[-2:]  # res.partner(23, 24)
partners[0] + partners[-2:]  # res.partner(41, 23, 24)

partners[0] | partners[-2:]  # res.partner(41, 23, 24)
partners[-2:] & partners[-1:]  # res.partner(24)
partners[-2:] - partners[-1:]  # res.partner(23)
```

It behaves just like former browse records, except that **updates are written in database**

```
partner = self.browse(5)

partner.name  # 'ETECH SA'
partner['name']  # 'ETECH SA'

partner.name = 'ETECH Group'  # Update database value
```

ENVIRONMENT

Environment encapsulates:

- **cr**: database cursor
- **uid**: id of current user
- **context**

```
cr, uid, context = self.env.args
self._cr  # <openerp.sql_db.Cursor object>
self._uid  # 1
self._context  # {'lang': 'fr_FR'}
```

Switching environments:

```
self.with_env(env2)  # self.env == env2
self.with_context(**new_context)  # self._context == new_context
self.sudo(new_user)  # self._uid == new_user.id
```

Environment also provides helpers:

```
self.env.ref('base.user_root')  # Resolve xml_id: res.users(1,)

self.env.user  # uid as a record

self.env['res.partner']  # Access to new-API model
```

API

Decorators to expose a new-style method to the old API

@api.model	cr, uid, *args, context
@api.multi	cr, uid, **ids**, *args, context
@api.one	cr, uid, ids, *args, context *as @api.multi but “autoloop” (one-by-one)*
@api.returns(‘self’)	*because new-style APIs tend to return recordsets and old-style APIs tend to return lists of ids*

```
@api.model
@api.returns('self', lambda value: value.id)
def create(self, values):
    …

@api.multi
def _do_something(self):
    for record in self:
       record…

@api.one
def _do_something(self):
    self…
```

API

Specific decorators

@api.onchange(fields)	recompute cache values if one of fields is updated in UI
@api.depends(fields)	recompute cache & database values if one of fields is updated
@api.constrains(fields)	check constraint if one of fields is updated

```
@api.onchange('partner_id')
def _onchange_dates(self):
    if self.partner_id:
        self.account_fiscal_position = self.partner_id.property_fiscal_account
    else:
        return {'warning': {
            'title': _('Warning!'),
            'message': _('Please select a partner!'),
        }, 'domain': {
            'partner_shipping_id': [('parent_id', '=', self.partner_id.id)],
        }}

@api.constrains('date_start', 'date_stop')
def _check_dates(self):
    if self.date_start > self.date_stop:
        raise Warning(_('End date must be posterior to Start date'))
```

METHODS

Generic accessor	**self.env[*object_name*]** may be used to obtain a model from any other *self.pool[object_name] # Deprecated::7.0*
@api.model def **search**(self, domain, offset=0, limit=None, order=None, count=False) => *Returns: list of records*	● **domain**: filter specifying search criteria ● **offset**: optional number of records to skip ● **limit**: optional max number of records to return ● **order**: optional columns to sort by (default: self._order) ● **count**: if True, returns only the number of records matching the criteria, not their ids
@api.model @api.returns('self', lambda value: value.id) def **create**(self, vals) => *Returns: the created record*	● **vals**: dictionary of field values
@api.multi def **write**(self, vals) => *Returns: True if records were updated*	● **vals**: dictionary of field values
@api.multi def **read**(self, fields=None, load='_classic_read') => *Returns: list of dictionaries with requested field values*	● **fields**: optional list of field names to return (default: all fields)

METHODS

@api.multi def **unlink**(self) *=> Returns: True if records were deleted*	
@api.model def **copy**(self, id, defaults=None) *=> Returns: id of duplicated record*	● **defaults**: dictionary of field values to modify in the copied values when creating the duplicated object
@api.model def **browse**(self, ids=None) *=> Returns: records as objects*	● **ids**: record ids
@api.model def default_get(self, fields) *=> Returns: a dictionary of the default values for fields (set on the object class, by the user preferences, or via the context)*	● **fields**: list of field names
@api.model def fields_get(self, fields=None) => *Returns:* a dictionary of field dictionaries, each one describing a field of the business object	● **fields**: list of field names

METHODS

@api.model def fields_view_get(self, view_id=None, view_type='form', toolbar=False) *=> Returns: dictionary describing the composition of the requested view (including inherited views)*	● **view_id**: id of the view or None ● **view_type**: type of view to return if view_id is None ('form','tree', …) ● **toolbar**: True to also return context actions
@api.model def **name_get**(self) *=> Returns: list of tuples with the text representation of requested objects for to-many relationships*	
@api.model def **name_search**(self, name='', args=None, operator='ilike', limit=100) *=> Returns: list of object names matching the criteria, used to provide completion for to-many relationships*	● **name**: object name to search for ● **operator**: operator for name criterion ● **domain, limit**: same as for search()
@api.multi def export_data(self, fields, raw_data=False) *=> Returns: list of object names matching the criteria, used to provide completion for to-many relationships*	● **fields**: list of field names ● **raw_data**: True to return value in native Python type

METHODS

@api.model
def load(self, fields, data)
=> Returns: {'ids': ids, 'messages': messages}
	● **fields**: list of field names
	● **data**: row-major matrix of data to import - *list(list(str))*

@api.multi def step_workflow(self) *=> Returns: reevaluate the workflow instances of the given record ids*	
@api.multi def **signal_workflow**(self, signal) *=> Returns: send given workflow signal and return a dict mapping ids to workflow results*	● **signal**: signal defined in workflow transition

METHODS

def mapped(self, func) => *Returns: result as a list or a recordset after applying `func` on all records*	● **func**: a function or a dot-separated sequence of field names
def filtered(self, func) => *Returns: recordset after filtering records with `func`*	● **func**: a function or a dot-separated sequence of field names
def sorted(self, key=None, reverse=False) => *Returns: recordset `self` ordered by `key`*	● **key**: either a function of one argument that returns a comparison key for each record ● **reverse**: if ``True``, return the result in reverse order

SEARCH DOMAIN

Search domain is a list of 3-tuples (field, operator, value) and operators '&' (and, explicite), '|' (or) and '!' (not), eg:

I search all sale order lines not done and lines done today
==
['|', ('state', 'not in', ('done', 'cancel')), '&', ('state', '=', 'done'), ('write_date', '>=', time.strftime('%Y-%m-%d'))]

Positive Operators

- String field: =, *in*, *like (case sensitive)*, *ilike (case insensitive)*
- Numeric field: =, >=, <=, >, <, <>
- Relational field: =, *in*, *child_of*

In Odoo, null value is equal to False. To know if a field *is set*, I search [(field, '!=', False)]
For *2many fields, when I search [(field, 'in', [2])], I search if [2] is in field values.

Polish notation

*"**Polish notation** [...] is a form of notation for logic, arithmetic, and algebra. Its distinguishing feature is that it **places operators to the left of their operands**. [...] it is readily parsed into abstract syntax trees"*, *source:* Wikipedia

Infix notation:
(A **AND** B) **OR** C
Polish notation:
OR, C, **AND**, B, A

EXCEPTIONS

Exception	Description
UserError(*msg*)	Warning = Alert pop-in
RedirectWarning(*msg, action_id, button_text*)	Warning with a possibility to redirect the user instead of simply diplaying the warning message
ValidationError(*msg*)	Violation of Python constraints
AccessError(*msg*)	Access rights error
MissingError(*msg*)	Missing record(s)
DeferredException(*msg, traceback*)	Exception object holding a traceback for asynchronous request. This class is used to store the possible exception occuring in the thread serving the first request, and is then sent to a polling request

CACHE MANAGEMENT

Record Cache

Odoo API works with batches (recordsets)

- Fields are computed in batches
- Reading records prefetches in batches

  Cache prefetching when accessing a field for the first time:
  - prefetches records browsed in the same environment
  - prefetches fields from the same table

```
# no database access
orders = env['sale.order'].browse(ids)         # ids

# no database access
order = orders[0]                              # ids

# prefetch order ids
partner = order.partner_id                     # data(ids), pids

# prefetch partner pids
name = partner.name                            # data(ids), data(pids)

# no database access
orders[1].partner_id.name                      # data(ids), data(pids)
```

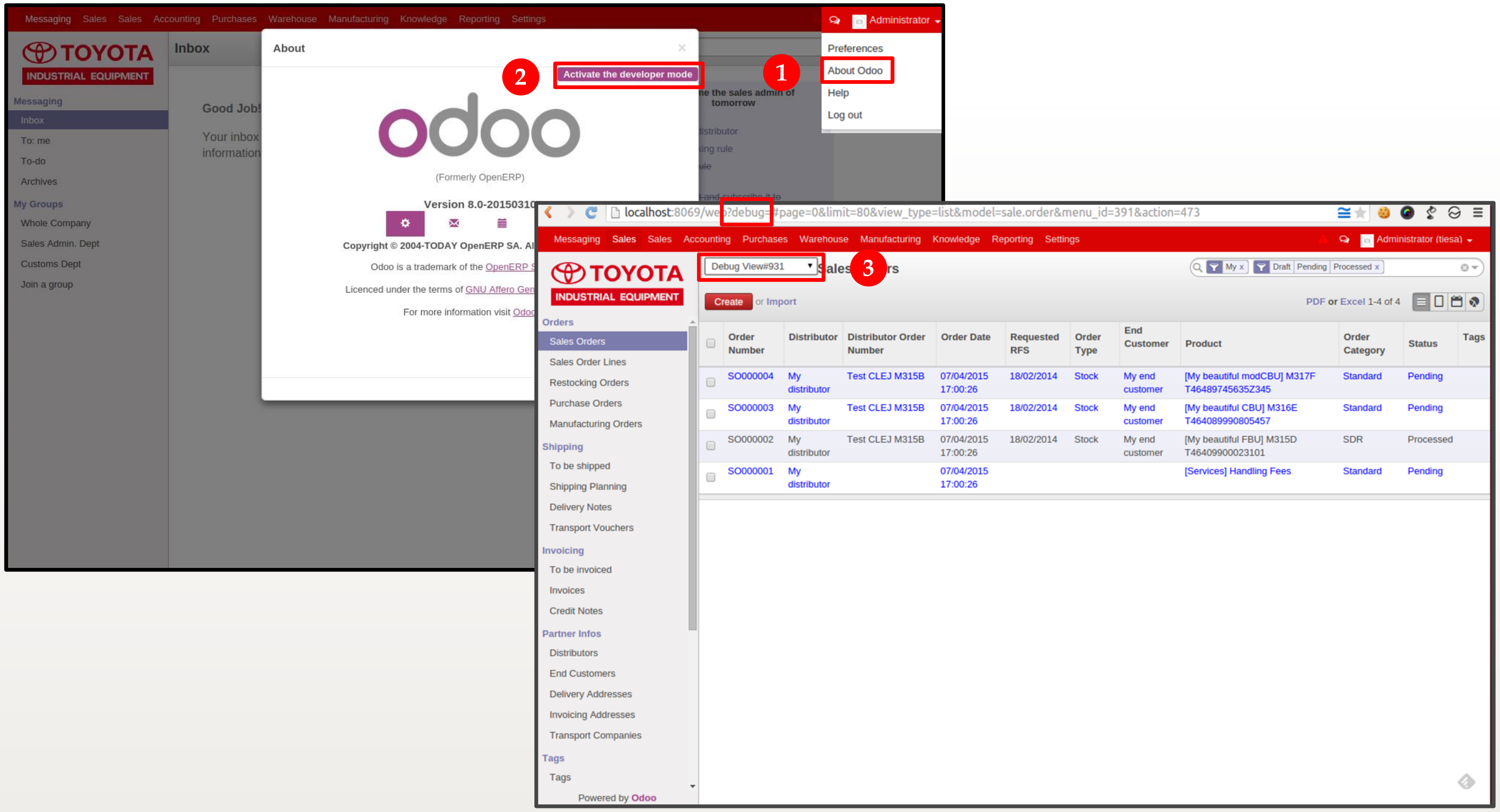
ORM Cache

For low cardinality tables with infrequent updates (*e.g. access rights, companies, menus, actions, views, bindings, decimal precisions, config parameters, modules, languages*), Odoo provides a LRU cache decorator for model methods.

```
from odoo import api, models, tools

class Lang(models.Model):

    @tools.ormcache('code')
    def _lang_get_id(self, code):
        return (self.search([('code', '=', code)]) or
                self.search([('code', '=', 'en_US')]) or
                self.search([], limit=1)).id

    @api.model
    def create(self, vals):
        self.clear_caches()  # Do idem for write and unlink methods
        return super(Lang, self).create(vals)
```

At each cache update, Odoo signals a registry change by incrementing a sequence in database. At each HTTP request, Odoo checks if the sequence has increased, and performs all necessary operations to update the registry. Using database sequence allows to support multi-workers and multi-instances.

DEVELOPER MODE <= 9.0



About

odoo

(Formerly OpenERP)

Version 8.0-20150310

Copyright © 2004-TODAY OpenERP SA. Al

Odoo is a trademark of the OpenERP S

Licenced under the terms of GNU Affero Gen

For more information visit Odoo

Good Job!

Your inbox information

localhost:8069/web?debug=#page=0&limit=80&view_type=list&model=sale.order&menu_id=391&action=473

Messaging Sales Sales Accounting Purchases Warehouse Manufacturing Knowledge Reporting Settings — Administrator (tiesa)

Sales Orders

Orders: Sales Orders, Sales Order Lines, Restocking Orders, Purchase Orders, Manufacturing Orders

Shipping: To be shipped, Shipping Planning, Delivery Notes, Transport Vouchers

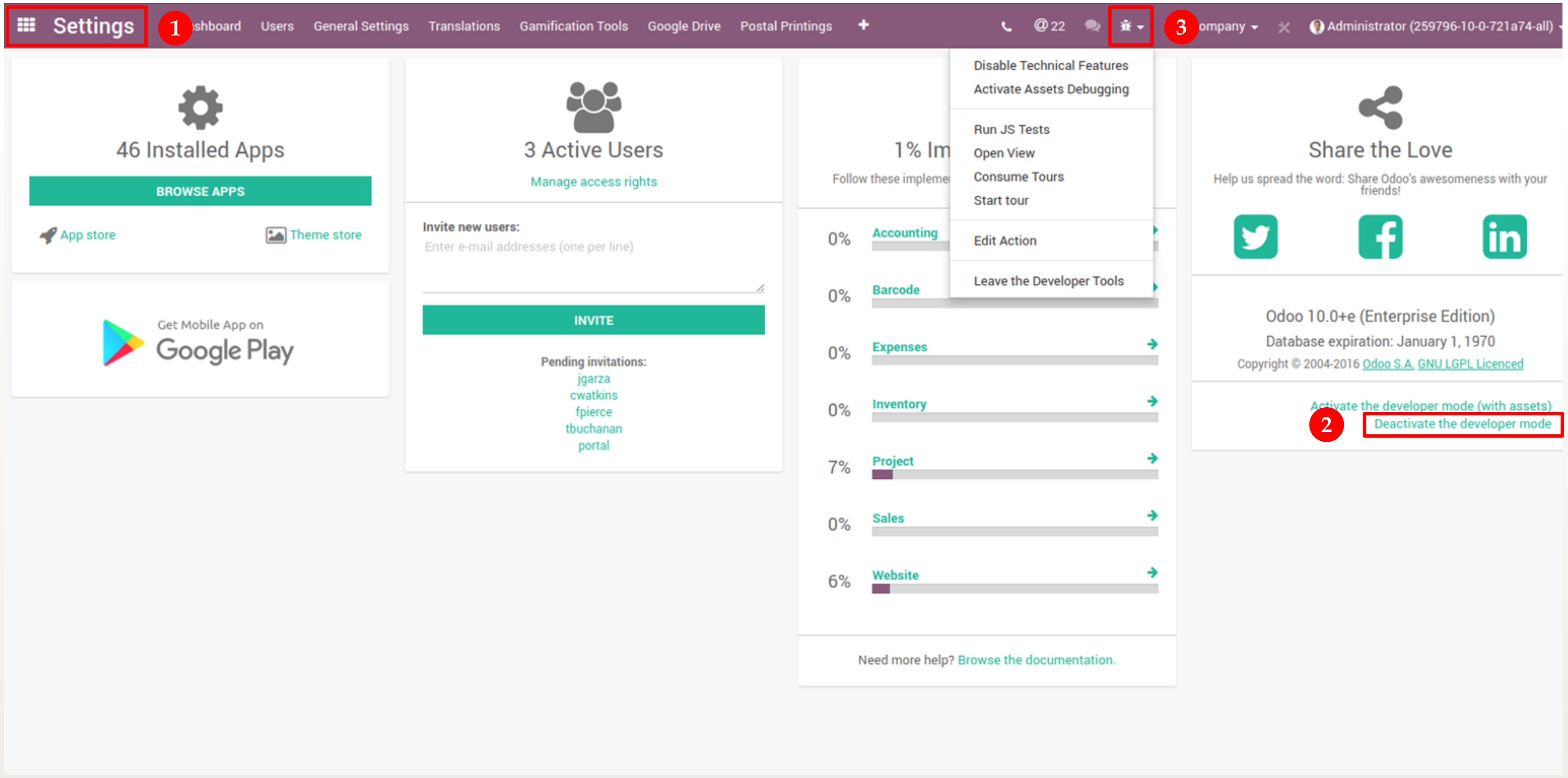
Invoicing: To be invoiced, Invoices, Credit Notes

Partner Infos: Distributors, End Customers, Delivery Addresses, Invoicing Addresses, Transport Companies

Tags: Tags

Order Number	Distributor	Distributor Order Number	Order Date	Requested RFS	Order Type	End Customer	Product	Order Category	Status	Tags
SO000004	My distributor	Test CLEJ M315B	07/04/2015 17:00:26	18/02/2014	Stock	My end customer	[My beautiful modCBU] M317F T464897456352345	Standard	Pending	
SO000003	My distributor	Test CLEJ M315B	07/04/2015 17:00:26	18/02/2014	Stock	My end customer	[My beautiful CBU] M316E T464089990805457	Standard	Pending	
SO000002	My distributor	Test CLEJ M315B	07/04/2015 17:00:26	18/02/2014	Stock	My end customer	[My beautiful FBU] M315D T464099000023101	SDR	Processed	
SO000001	My distributor		07/04/2015 17:00:26				[Services] Handling Fees	Standard	Pending	

DEVELOPER MODE >= 10.0



DATA IMPORT

Data can be import via **XML**, CSV, SQL, YML files.

```
<?xml version="1.0" encoding="utf-8"?>
<odoo noupdate="1">

   <record model="object_model_name" id="object_xml_id">
     <field name="field1">value1</field>
     <field name="field2">value2</field>
   </record>

   <record model="object_model_name2" id="object_xml_id2">
     <field name="field1" ref="module.object_xml_id"/>
     <field name="field1" eval="ref('module.object_xml_id')"/>
   </record>

</odoo>
```

id	the unique (per module) external identifier of this record (xml_id)
ref	may be used instead of normal element content to reference another record (works cross-module by prepending the module name)
eval	used instead of element content to provide value as a Python expression, that can use the ref() method to find the database id for a given xml_id
noupdate	if True, don't update records if already imported

DATA IMPORT

Data can be import via XML, CSV, SQL, YML files.

⚠ noupdate="0"

```
ir.model.access.csv
"id","name","model_id:id","group_id:id","perm_read","perm_write","perm_create","perm_unlink"
"access_idea_idea","idea.idea","model_idea_idea","base.group_user",1,0,0,0
"access_idea_vote","idea.vote","model_idea_vote","base.group_user",1,0,0,0
```

file name	the name of file must be the name of my model with the extension .csv
id	column containing identifiers for relationships (xml_id)
many2one_field	reconnect many2one using name_search()
many2one_field:id **many2one_field/id**	reconnect many2one based on object's xml_id
many2one_field.id	reconnect many2one based on object's database id
many2many_field	reconnect via name_search(), multiple values with commas
many2many_field:id	reconnect w/ object's xml_id, multiple values with commas
many2many_field.id	reconnect w/ object's database id, multiple values with commas
one2many_field/field	creates one2many destination record and sets field value one row by destination record

MENUS & ACTIONS

Actions are declared as regular records and can be triggered in 3 ways:

- by clicking on menu items linked to a specific action
- by clicking on buttons in views, if these are connected to actions
- as contextual actions on an object or objects list, visible in "More" or "Print" menu, via Action Bindings (*ir.values to Odoo v10, ir.binding from Odoo v11*)

Types of actions

- **Window** (ir.actions.act_window): open model's views
- URL (ir.actions.act_url): open a external URL
- Client (ir.actions.act_client): open javascript view
- Server (ir.actions.server): can
  - Send email
  - Trigger a workflow signal
  - Execute Python code
  - Run a client action
  - Create or copy a new record
  - Write on a record
  - Execute several actions
- Report (ir.actions.report.xml): generate a report

Automation

- Automated Actions (*ir.actions.server*): triggered on server events
- Scheduled Actions (*ir.cron*): triggered on datetime, based on an internal scheduler

MENUS & ACTIONS

```
<record model="ir.actions.act_window" id="action_id">
  <field name="name">action.name</field>
  <field name="view_id" ref="view_id"/>
  <field name="domain">[list of 3-tuples (max 250 characters)]</field>
  <field name="context">{context dictionary (max 250 characters)}</field>
  <field name="res_model">object.model.name</field>
  <field name="view_type">form|tree</field>
  <field name="view_mode">form,tree,calendar,graph,kanban,gantt</field>
  <field name="target">new</field>
  <field name="search_view_id" ref="search_view_id"/>
</record>
```

id	identifier of the action in table ir.actions.act_window, must be unique name action name (required)
view_id	specific view to open (if missing, highest priority view of given type is used)
domain	tuple (see search() arguments) for filtering the content of the view context
context	dictionary to pass to the view
res_model	object model on which the view to open is defined
view_type	set to form to open records in edit mode, set to tree for a hierarchy view only
view_mode	if view_type is form, list allowed modes for viewing records (form, tree, ...)
target	set to new to open the view in a new window/popup

MENUS & ACTIONS

```
<menuitem id="menu_id" parent="parent_menu_id" name="label"
 action="action_id" groups="group_name1,group_name2" sequence="15"/>
```

The menuitem element is a shortcut for declaring an ir.ui.menu record and connect it with a corresponding action via an ir.model.data record.

VIEWS & INHERITANCE

Views form a hierarchy. Several views of the same type can be declared on the same object, and will be used depending on their priorities. By declaring an inherited view it is possible to add/remove features in a view.

```
<record model="ir.ui.view" id="view_id">
  <field name="name">view.name</field>
  <field name="model">object.name</field>
  <field name="priority" eval="16"/>
  <field name="groups_id" eval="[(6, 0, [ref('base.group_no_one')])]"/>
  <field name="arch" type="xml">
  <!-- view content: <form>,<tree>,<search>,<graph>,<pivot>,<kanban>,<calendar>,<gantt>,<grid> -->
  </field>
</record>
```

id	unique view identifier
model	object model on which the view is defined (same as res_model in actions)
inherit_id	inherited view
mode	primary (default if `inherit_id` is unset) or extension (default if `inherit_id` is set)
priority	view priority, smaller is higher (default: 16)
groups_id	groups allowed to view / use the current view
arch	architecture of the view, see various view types below

Note: You can force the use of some views by declaring window action views (ir.actions.atc_window.view) or by passing in context {'%s_view_ref' % view_type: view_id}.

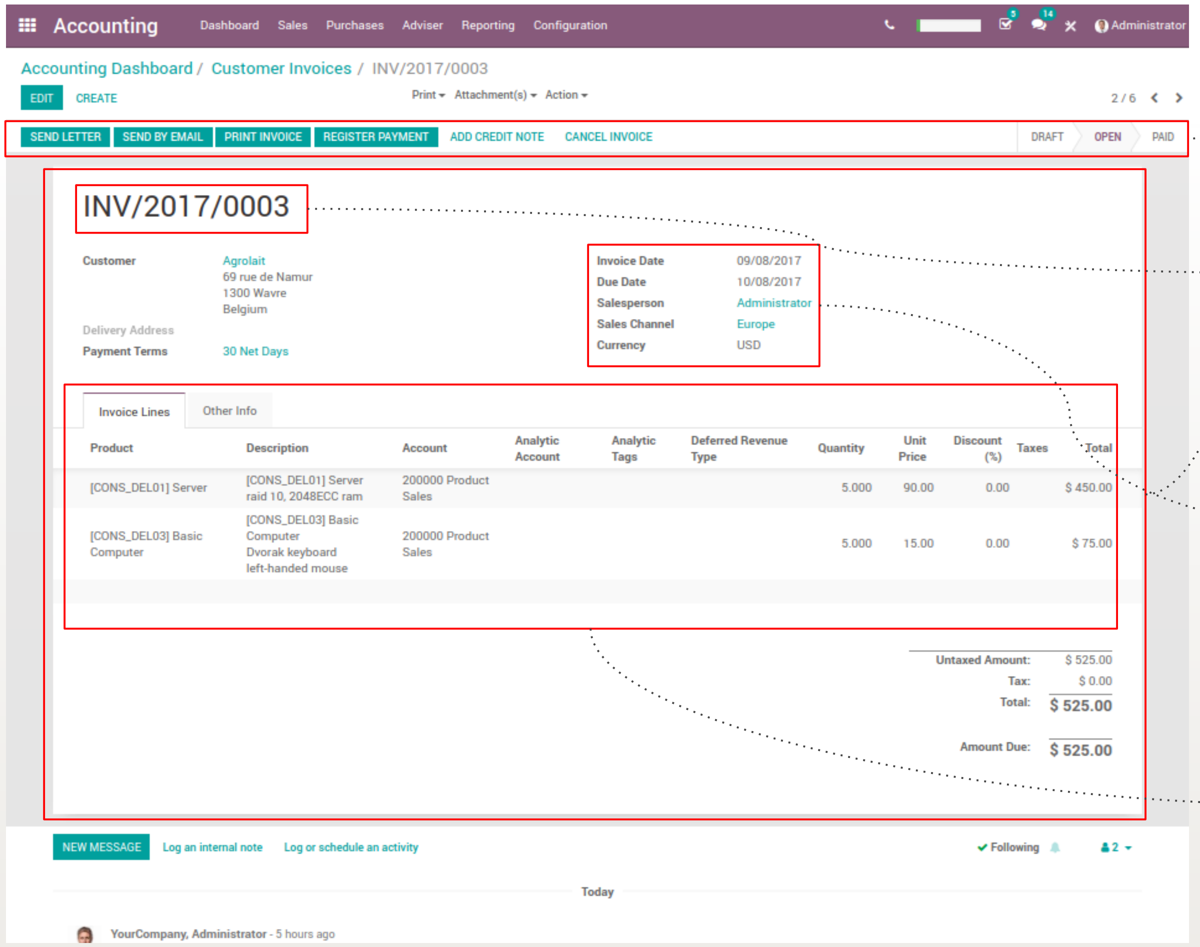
VIEWS & INHERITANCE

Existing views should be modifying through inherited views, never directly. An inherited view references its parent view using the inherit_id field, and may add or modify existing elements in the view by referencing them through XPath expressions, and specifying the appropriate position:

- inside: put inside match (default)
- replace: replace match
- before: put before match
- after: put after match
- attributes: override match attributes like invisible / readonly / attrs / string / ...

```
<record model="ir.ui.view" id="view2_id">
  <field name="name">view2.name</field>
  <field name="model">object.name</field>
  <field name="inherit_id" ref="view_id"/>
  <field name="priority" eval="16"/>
  <field name="arch" type="xml">
    <field name="field_name" position="attributes">
      <attribute name="invisible">1</attribute>
      <attribute name="string">New label</attribute>
    </field>
    <xpath expr="//field[@name='field_name']" position="after"> # depricated
    <field name="field_name" position="after">
      <field name="field2_name"/>
    </xpath>
  </field>
</record>
```

STRUCTURAL COMPONENTS



Accounting Dashboard / Customer Invoices / INV/2017/0003

INV/2017/0003

Customer	Agrolait 69 rue de Namur 1300 Wavre Belgium
Delivery Address	
Payment Terms	30 Net Days

Invoice Date	09/08/2017
Due Date	10/08/2017
Salesperson	Administrator
Sales Channel	Europe
Currency	USD

Invoice Lines | Other Info

Product	Description	Account	Analytic Account	Analytic Tags	Deferred Revenue Type	Quantity	Unit Price	Discount (%)	Taxes	Total
[CONS_DEL01] Server	[CONS_DEL01] Server raid 10, 2048ECC ram	200000 Product Sales				5.000	90.00	0.00		$ 450.00
[CONS_DEL03] Basic Computer	[CONS_DEL03] Basic Computer Dvorak keyboard left-handed mouse	200000 Product Sales				5.000	15.00	0.00		$ 75.00

Untaxed Amount:	$ 525.00
Tax:	$ 0.00
Total:	$ 525.00
Amount Due:	$ 525.00

YourCompany, Administrator - 5 hours ago

`<header/>`

`<sheet/>`

`<div/>` *(col=1 by default) Field labels are not displayed. To display label, add <label for="field_name"/>*

`<group/>` *(col=2 by default) Field labels are displayed*
To hide field label, add attribute <field name="field_name" nolabel="1"/>

<notebook>
  <page string "Invoice Lines"/>
  <page string="Other Info"/>
</notebook>

SEMANTIC COMPONENTS

<button/>

- icon & string: button text if no icon else alt text
- type: workflow (default) / object / action
- name: workflow signal / method name / action id
- confirm: confirmation message to display (and for the user to accept) before the button calling
- context: merged into the view’s context when performing the button call

<field/>

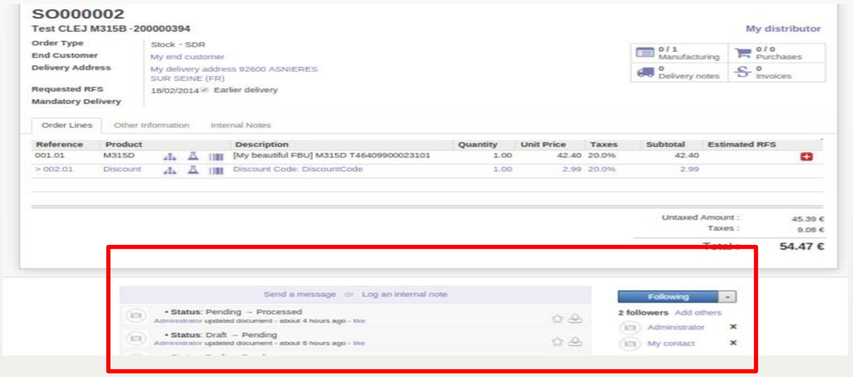
- name: technical field name. A given name can only use once per view
- string: field title / label
- widget & options: look & feel
- sum / avg: displays the corresponding aggregate at the bottom of the column
- domain: for relational fields only, filters to apply when displaying existing records for selection
- placeholder: help message to display in empty fields. Can replace field labels in complex forms
- help: tooltip displayed for users when hovering the field or its label
- password: indicates that a Char field stores a password and that its data shouldn't be displayed

For all *structural and semantic* components

- invisible / readonly / required
- attrs: dynamic attributes based on record values, e.g. attrs=”{‘invisible’: [(‘state’, ‘=’, ‘draft’)], ‘required’: [(‘state’, ‘=’, ‘done’)], ‘readonly’: [(‘state’, ‘=’, ‘cancel’)]}”
- states: visible only if state matches
- groups: lists the groups which should be able to see the item
- class: Odoo/Bootstrap class to set on the generated element

MESSAGING FEATURES

Once you've added chatter support on your model, users can easily add messages or internal notes (mail.message) on any record of your model; every one of those will send a notification: to all followers for messages, to employee (*base.group_user*) users for internal notes. If your mail gateway and catchall address are correctly configured, these notifications will be sent by e-mail and can be replied-to directly from your mail client; the automatic routing system will route the answer to the correct thread.



SO000002
Test CLEJ M315B -200000394

My distributor

Order Type	Stock - SDR
End Customer	My end customer
Delivery Address	My delivery address 92600 ASNIERES SUR SEINE (FR)
Requested RFS	18/02/2014 ✓ Earlier delivery
Mandatory Delivery	

0 / 1 Manufacturing · 0 / 0 Purchases · 0 Delivery notes · 0 Invoices

Order Lines · Other Information · Internal Notes

Reference	Product	Description	Quantity	Unit Price	Taxes	Subtotal	Estimated RFS
001.01	M315D	[My beautiful FBU] M315D T464099000023101	1.00	42.40	20.0%	42.40	
> 002.01	Discount	Discount Code: DiscountCode	1.00	2.99	20.0%	2.99	

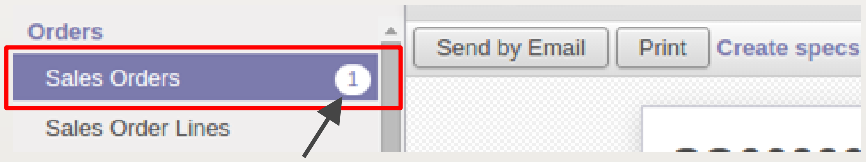
Untaxed Amount : 45.39 €
Taxes : 9.08 €
Total : 54.47 €

Send a message or Log an internal note

• Status: Pending → Processed
Administrator updated document - about 4 hours ago - like

• Status: Draft → Pending
Administrator updated document - about 6 hours ago - like

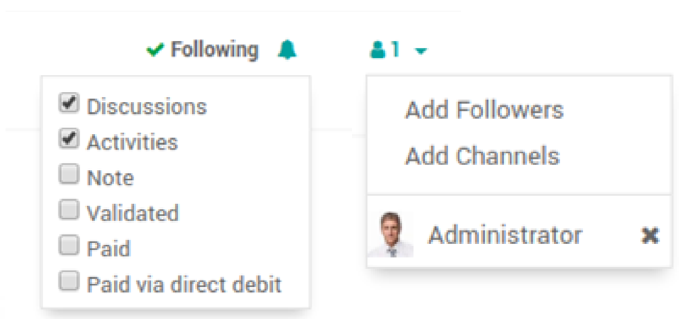
Following
2 followers Add others
Administrator
My contact



= 1 sale order with new message(s)

```
class MyModel(models.Model):
    _name = 'my.model'
    _inherit = ['mail.thread']


<!-- In form view -->
<div class="oe_chatter">
  <field name="message_follower_ids"
widget="mail_followers" widget="base.group_user"/>
  <field name="message_ids" widget="mail_thread"/>
</div>


class MyModel(models.Model):
    _name = 'my.model'
    _inherit = ['ir.needaction_mixin']
```

MESSAGING FEATURES



Subtypes (mail.message.subtype) give you more granular control over messages. Subtypes act as a classification system for notifications, allowing subscribers to a document to customize the subtype of notifications they wish to receive.

```
<record model="mail.message.subtype" id="account.mt_invoice_paid">
  <field name="name">Paid</field>
  <field name="res_model">account.invoice</field>
  <field name="default" eval="False"/>
  <field name="description">Invoice paid</field>
</record>
```

The mail module adds a tracking system on fields, allowing you to log changes to specific fields in the record's chatter. To add tracking to a field, simple add the track_visibility attribute with the value *onchange* (if it should be displayed in the notification only if the field changed) or *always* (if the value should always be displayed in change notifications even if this particular field did not change - useful to make notification more explanatory by always adding the name field, for example).

```
class AccountInvoice(models.Model)
    _name = 'account.invoice'

    type = fields.Selection(..., track_visibility='always')
    state = fields.Selection(..., track_visibility='onchange')
```

MESSAGING FEATURES

Aliases (mail.alias) are configurable email addresses that are linked to a specific record (which usually inherits the mail.alias.mixin model) that will create new records when contacted via e-mail. They are an easy way to make your system accessible from the outside, allowing users or customers to quickly create records in your database without needing to connect to Odoo directly. A single incoming gateway can be used by many aliases.

```
   <record model="mail.message.subtype" id="hr_expense.mail_alais_expense">
     <field name="alias_name">expense</field>
     <field name="alias_model_id" ref="hr_expense.model_hr_expense"/>
     <field name="alias_user_id" ref="base.user_root"/>
     <field name="alias_contact">employees</field><!-- Policy to post a message: everyone,
partners, followers -->
   </record>
```

Aliases (mail.alias.mixin) are usually configured on a parent model which will then create specific record when contacted by e-mail. For example, Project have aliases to create tasks or issues, Sales Team have aliases to generate Leads, Job Position to link new Applicants.

```
class HrJob(models.Model)
    _name = 'hr.job'
    _inherit = ['mail.alias.mixin', 'hr.job']

    alias_id = fields.Many2one('mail.alias', "Alias", required=True, ondelete='restrict')
```

REPORT (QWEB)

Documentation: https://www.odoo.com/documentation/10.0/reference/reports.html

Reports are web pages, written in HTML/QWeb, like all regular views in Odoo. You can use the usual QWeb control flow tools. The PDF rendering itself is performed by wkhtmltopdf v0.12.1.

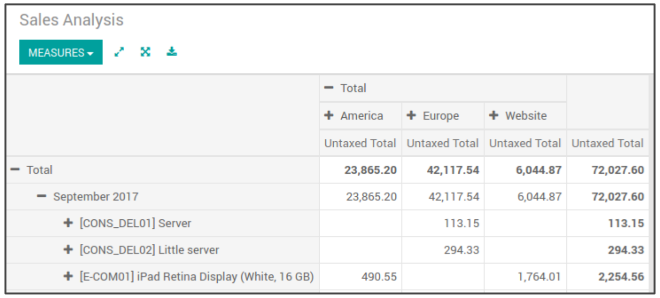
If you want to create a report on a certain model, you will need to define this Report and the Report template it will use. If you wish, you can also specify a specific Paper Format for this report. Finally, if you need access to more than your model, you can define a Custom Reports class that gives you access to more models and records in the template.

Reports are dynamically generated by the report module and can be accessed directly via URL:

- http://<server-address>/report/html/sale.report_saleorder/38
- http://<server-address>/report/pdf/sale.report_saleorder/38

```
   <template id="account.report_invoice">
     <t t-call="report.hmlt_container">
       <t t-foreach="docs" t-as="o"><!-- docs = records for the current report -->
         <t t-call="report.external_layout">
           <div class="page">
            <h2>Report title</h2>
            <p>This object's name is <span t-field="o.name"/></p>
            <p t-if="o.type.startswith('out')" t-att-class="'refund' in o.type and 'bg-danger'
or ''"><span t-esc="o.type == 'refund' and 'Refund' or 'Invoice'"/></p>
           </div>
         </t>
       </t>
     </t>
   </template>
```

REPORT (SQL VIEW)



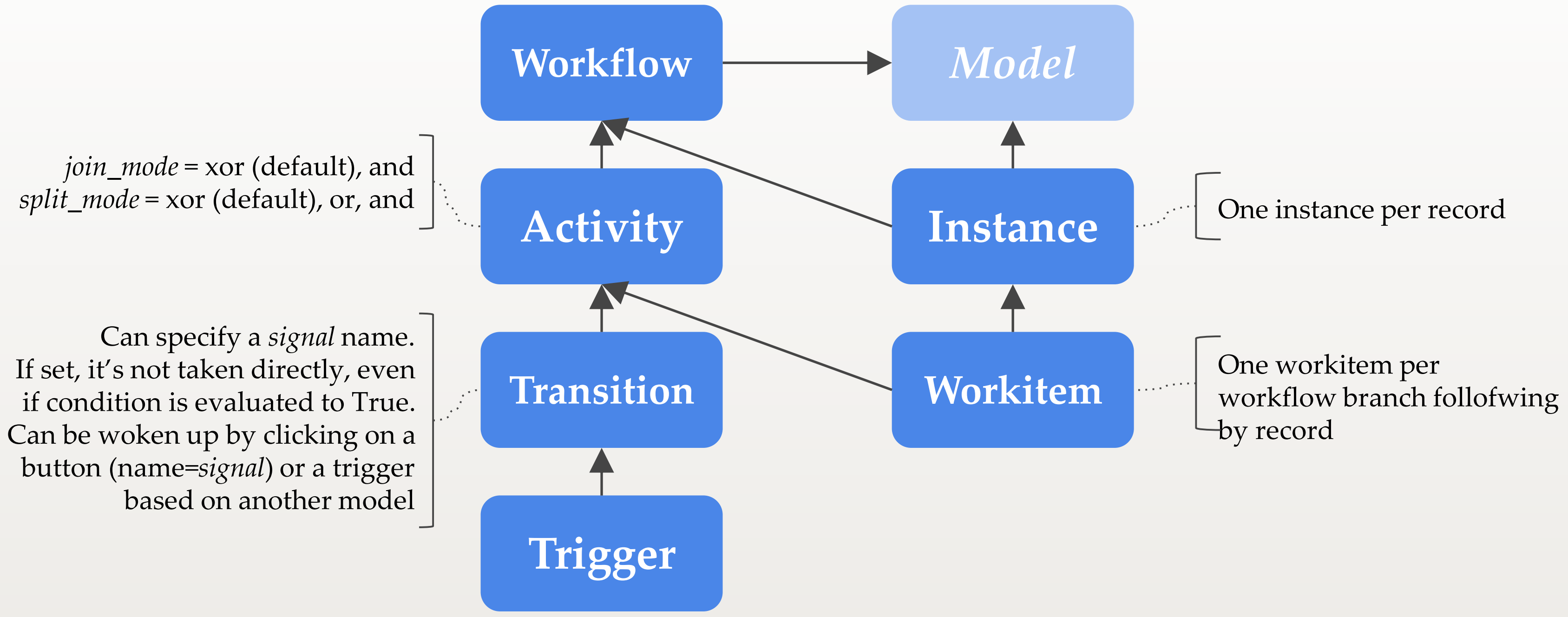
	− Total			
	+ America	+ Europe	+ Website	
	Untaxed Total	Untaxed Total	Untaxed Total	Untaxed Total
− Total	**23,865.20**	**42,117.54**	**6,044.87**	**72,027.60**
− September 2017	23,865.20	42,117.54	6,044.87	**72,027.60**
+ [CONS_DEL01] Server		113.15		**113.15**
+ [CONS_DEL02] Little server		294.33		**294.33**
+ [E-COM01] iPad Retina Display (White, 16 GB)	490.55		1,764.01	**2,254.56**

Odoo provides Pivot view to facilitate data analysis. Like any view, this one is based on a unique model. So if you want to cross dimensions / data from several models, you need to create a new model based on a SQL view.

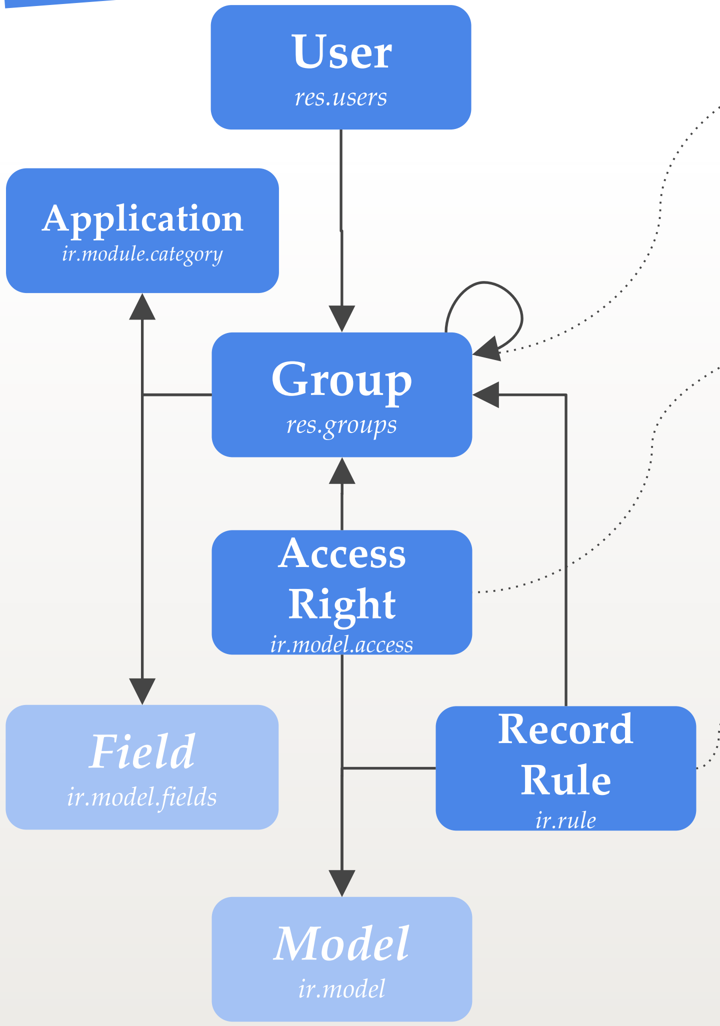
```
from odoo import tools
from odoo import api, fields, models


class SaleReport(models.Model):
    _name = "sale.report"
    _description = "Sales Orders Statistics"
    _auto = False

    name = fields.Char('Order Reference', readonly=True)
    date = fields.Datetime('Date Order', readonly=True)
    ...

    @api.model_cr
    def init(self):
        tools.drop_view_if_exists(self.env.cr, self._table)
        self.env.cr.execute("CREATE or REPLACE VIEW %s as (%s)" % (self._table, subqueries))
```

WORKFLOW

Workflows may be associated with any object in Odoo, and are entirely customizable. Workflows are used to structure and manage the life-cycles of business objects and documents, and define transitions, triggers, etc. with graphical tools.
Workflows, activities (nodes or actions) and transitions (conditions) are declared as XML records, as usual. The tokens that navigate in workflows are called workitems.


Workflow
Model
join_mode = xor (default), and
split_mode = xor (default), or, and
Activity
Instance
One instance per record
Can specify a *signal* name.
If set, it's not taken directly, even if condition is evaluated to True.
Can be woken up by clicking on a button (name=*signal*) or a trigger based on another model
Transition
Workitem
One workitem per workflow branch follofwing by record
Trigger

SECURITY



Users added to a group are automatically added in the inherited groups.
If the groups of an application are hierarchical, they appear as a selection field, otherwise as checkboxes.

Available permissions are :

- creation (`perm_create`),
- searching and reading (`perm_read`),
- updating existing records (`perm_write`)
- deleting existing records (`perm_unlink`)

A record rule has:

- a model on which it applies
- a set of permissions to which it applies (e.g. if `perm_read` is set, the rule will only be checked when reading a record)
- a set of user groups to which the rule applies, if no group is specified the rule is *global*
- a *domain* used to check whether a given record matches the rule (and is accessible) or does not (and is not accessible). The domain is evaluated with two variables in context: `user` is the current user's record and `time` is the time module

Interaction between rules: GLOBAL_RULE_1 AND GLOBAL_RULE_2 AND ((GROUP_A_RULE_1 OR GROUP_A_RULE_2) OR (GROUP_B_RULE_1 OR GROUP_B_RULE_2))

TRANSLATIONS

Odoo provides support for translating interface and data using .po files.

To customize translations, I invite you to export terms to translate for a given module and a given language. Menu: Settings > Translations > Import/Export > Export translations

This gives you a file called *language*.po which should be moved to the *your_module/i18n/* directory.

Implicitely, the content of field/label string / action/menu name / help / sum/avg / confirm / placeholder are translatable.

```
from odoo import _

# bad, the extract may work but it will not translate the text correctly
_("Scheduled meeting with %s" % invitee.name)

# good
_("Scheduled meeting with %s") % invitee.name
```

TESTS (UNIT TESTS)

Odoo unit tests

- XML (from Odoo 5.0)
- YAML (from Odoo 6.0)
- Python (from Odoo 7.0)

To write a test in Python, simply define a `tests` sub-package in your module, it will be automatically inspected for test modules. Test modules should have a name starting with `test_` and should be imported from `tests/__init__.py`.

```
from odoo.tests.common import TransactionCase


class TestSaleOrder(TransactionCase):

    def setUp(self):
        super(TestSaleOrder, self).setUp()
        self._sales_person = self.env.ref(
            'sales_person')

    def test_10_sale_order_draft(self):
        """
        1. As Sales person, I create a quotation
        2. I check that it is in state 'draft'
        """
        vals = {...}
        order = self.env['sale.order']. \
            sudo(self._sales_person).create(vals)
        self.assertEquals(u'draft', order.state,
            'Sale order is not draft!')
```

Odoo provides a number of utilities and helpers related to testing Odoo content (modules, mainly):

- `TransactionCase`: each test method is run in its own transaction, and with its own cursor. The transaction is rolled back and the cursor is closed after each test method
- `SingleTransactionCase`: all test methods are run in the same transaction, the transaction is started with the first test method and rolled back at the end of the last

Note :

- *setUp / tearDown is called before / after each test method*
- *setUpClass / tearDownClass is called at the beginning / end of test class*

TESTS (UNIT TESTS)

Allows to **test flows**

To be sure your tests will be executed in order, name your tests using the following pattern: test_10_xxx, test_20_yyy…

The most important part of a test is its docstring. The description of a test must:

- be understandable by the customer, so written in the customer language
- describe for each step:
    - user profile
    - data
    - expected result
    - checking

1. As sales person, I create a quotation for the customer Agrolait including a white iPad for a unit price of 800 €
2. I check that the total amount is 800 € since the taxes are included in price
3. I check that the document is in the status "Quotation"
4. Always as sales person, I click on the button "Confirm sale order"
5. I check that the document is in the status "Sale Order"

…

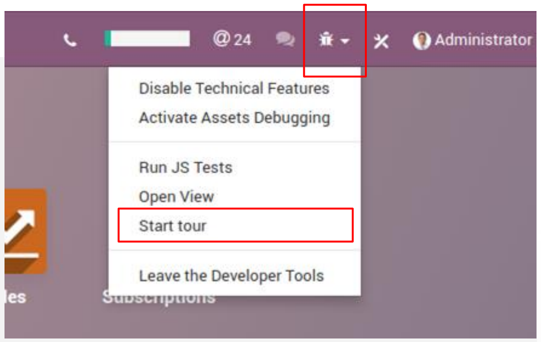
TESTS (UNIT TESTS)

Helper	Check if
self.assertEqual(a, b) / self.assertNotEqual(a, b)	a == b / a != b
self.assertTrue(a) / self.assertFalse(a)	bool(a) is True / bool(a) is False
self.assertIn(a, b) / self.assertNotIn(a, b)	a in b / a not in b
self.assertIsInstance(a, b) / self.assertIsNotInstance(a, b)	isinstance(a, b) / not isinstance(a, b)
self.assertIsNone(a) / self.assertIsNotNone(a)	a is None / a is not None

with self.assertRaises(*ExceptionType*):
    your_test
	an exception of type *ExceptionType* is raised when you execute your test

TESTS (JS TOURS)

Allows to **test flows and UI**
Available in the backend, the website, the POS, iframes…


@24
Administrator
Disable Technical Features
Activate Assets Debugging
Run JS Tests
Open View
Start tour
Leave the Developer Tools
Subscriptions

```
odoo.define('project.tour', function(require) {
    "use strict";

    var core = require('web.core');
    var tour = require('web_tour.tour');
    var _t = core._t;

    var options = {
        url: "/web",
    };
    var steps = [{
        trigger: '.o_app[data-menu-xmlid="sales_team.menu_base_partner"]',
        content: _t("Ready to boost your sales?"),
        position: 'bottom',
    },];  // Array of steps

    tour.register('project_tour', options, steps);
});
```

```
from odoo.tests import common, HttpCase

@common.at_install(False)
@common.post_install(True)
class TestUi(HttpCase):

    def test_01_project_tour(self):  # Execute automatically a JS tour with unittests
        self.phantom_js("/web",
                        "odoo.__DEBUG__.services['web_tour.tour'].run('project_tour')",
                        "odoo.__DEBUG__.services['web_tour.tour'].tours.project_tour.ready",
                        login="admin")
```

TESTS (JS TOURS)

A **step** is an object with the following keys:

- trigger (mandatory): css selector of the element on which to attach the tip
- extra_trigger: additional css selector that must match a DOM element for the tip to be displayed
- content: the html content of the tip
- position: right (default), left, top or bottom
- width: default: 270px
- run: the operation to perform
    - text(*value*): writes value in the element (handles select elements as well)
    - click: clicks on the element
    - drag_and_drop(*to*): drags and drops the element inside the element matching the given css selector
    - keydown(*keycodes*):simulates a keydown event on the element for the given keycodes
    - auto: default, calls click or text according to the element's type

There are the different **options**:

- url: the url to load before starting the tour
- skip_enabled: set to *true* to add a link to skip the tour in its tips
- test: set to *true* if the tour is dedicated to tests
- wait_for: a deferred that must be resolved for the tour to be read

TESTS (JS TOURS)

Helper to write your selectors	Description
:containsExact(*value*)	matches if the content of the element is exactly value, e.g. ‘span:containsExact(OdooExperience)’ ✓ <span>OdooExperience</span> ✗ <span>OdooExperience 2016</span>
:containsExactCase(*value*)	like containsExact but case sensitive
:containsRegex(*value*)	like containsExact but with a regular expression
:propValueContains(*value*)	matches if the element’s value property contains to value

<u>Tip to select a menu</u>

✗ Avoid poor selectors like *‘.o_app:nth-child(2)’* or *‘.o_app:contains(”CRM”)’*
✓ Write a selector based on the xml_id *‘.o_app[data-menu-xmlid=”sales_team.menu_base_partner”]’*

By default, menu xml_ids are not loaded in the webclient ⇒ Set the flag load_xmlid to *True* :

```
<record id="sales_team.menu_base_partner" model="ir.ui.menu">
  <field name="load_xmlid" eval="True"/>
</record>
```

WEB SERVICES (XML/RPC)

Documentation:
https://www.odoo.com/documentation/10.0/api_integration.html

```
python
>>> import xmlrpclib
>>> def xmlrpc(service, method, *args):
...     socket = xmlrpclib.ServerProxy('http://localhost:8069/xmlrpc/2/' + service)
...     return getattr(socket, method)(*args)

# Get database
>>> dbname = xmlrpc('db', 'list')[0]

# Authenticate
>>> login = 'admin'
>>> password = 'admin'
>>> uid = xmlrpc('common', 'login', dbname, login, password)

# Search draft invoices and validate them
# xmlrpc('object', 'execute', dbname, uid, password, model, method, *args)
# xmlrpc('object', 'execute_kw', dbname, uid, password, model, method, args)
>>> invoice_ids = xmlrpc('object', 'execute', dbname, uid, password, 'account.invoice',
'search', [('state', '=', 'draft')])
>>> xmlrpc('object', 'execute', dbname, uid, password, 'account.invoice', 'action_invoice_open',
invoice_ids)

# Print consolidated report
>>> report_data = xmlrpc('report', 'render_report', dbname, uid, password,
'account.report_invoice', invoice_ids)['result'].decode('base64')
```

WEB SERVICES (JSON/RPC)

Adapted to mobile interfaces (because JSON)

```
python
>>> import json
>>> import random
>>> import requests
>>> def jsonrpc(service, method, params, session_id=None):
...     headers = {'Content-Type': 'application/json', 'Accept': 'application/json'}
...     payload = {'jsonrpc': '2.0', 'method': 'call', 'params': params, 'id': random.randint(0,
1000000000)}
...     response = requests.post('http://localhost:8069/web/%s/%s' % (service, method),
data=json.dumps(payload), headers=headers, cookies={'session_id': session_id}).json()
...     if response.get('error'):
...         raise Exception(response['error'])
...     return response['result']

# Get database
>>> dbname = jsonrpc('database', 'list', {})[0]

# Authenticate
>>> login = 'admin'
>>> password = 'admin'
>>> session_id = jsonrpc('session', 'authenticate', {'db': dbname, 'login': login, 'password':
password})['session_id']

# Search draft invoices and validate them
>>> invoice_ids = [r['id'] for r in jsonrpc('dataset', 'search_read', {'model': 'account.invoice',
'domain': [('state', '=', 'draft')], 'fields': ['id']}, session_id)['records']]
>>> jsonrpc('dataset', 'call_kw', {'model': 'account.invoice', 'method': 'action_invoice_open', 'args':
[invoice_ids], 'kwargs': {}}, session_id)
```

WEB CONTROLLERS

```
from odoo import http

class Academy(http.Controller):

    # A website menu points to a route or an external url
    @http.route('/academy/academy/', auth='public', website=True)
    def index(self, **kw):
        Teachers = http.request.env['academy.teachers']
        return http.request.render('academy.index', {
            'teachers': Teachers.search([])
        })

    @http.route('/academy/<int:id>/', auth='public', website=True)
    def academy(self, id):
        return '<h1>{} ({})</h1>'.format(id, type(id).__name__)

    # auth='user' means this page is accessible only for connected users
    @http.route('/academy/<model("academy.teachers"):teacher>/', auth='user', website=True)
    def teacher(self, teacher):
        return http.request.render('academy.biography', {
            'person': teacher
        })

class MailChatController(BusController):

    @route('/mail/chat_post', type="json", auth="none", methods=['POST'])
    def mail_chat_post(self, uuid, message_content, **kwargs):
        …
        return message and message.id or False
```

WEB CONTROLLERS

```
<!-- Templates used by previous routes -->
<?xml version="1.0" encoding="utf-8"?>
<odoo>
    <template id="academy.index">
        <t t-call="website.layout">
            <t t-set="title">Academy</t>
            <div class="oe_structure">
                <div class="container">
                    <t t-foreach="teachers" t-as="teacher">
                        <p><a t-attf-href="/academy/{{ slug(teacher) }}">
                          <t t-esc="teacher.name"/></a>
                        </p>
                    </t>
                </div>
            </div>
        </t>
    </template>
    <template id="academy.biography">
        <t t-call="website.layout">
            <t t-set="title">Academy</t>
            <div class="oe_structure"/>
            <div class="oe_structure">
                <div class="container">
                    <p><t t-esc="person.id"/> <t t-esc="person.name"/></p>
                </div>
            </div>
            <div class="oe_structure"/>
        </t>
    </template>
</odoo>
```

JAVASCRIPT

Odoo Framework based on:
JQuery, Underscore.js

TODO

https://www.odoo.com/documentation/10.0/reference/javascript.html
https://www.odoo.com/documentation/10.0/reference/mobile.html
https://www.odoo.com/documentation/10.0/howtos/web.html

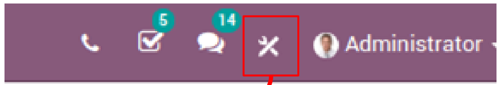
WEBSITE THEMES & SNIPPETS

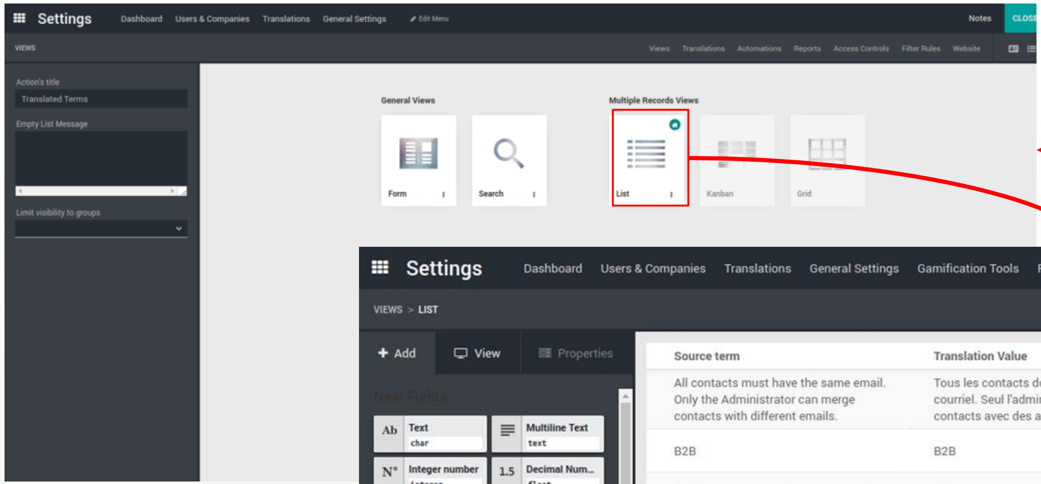
TODO

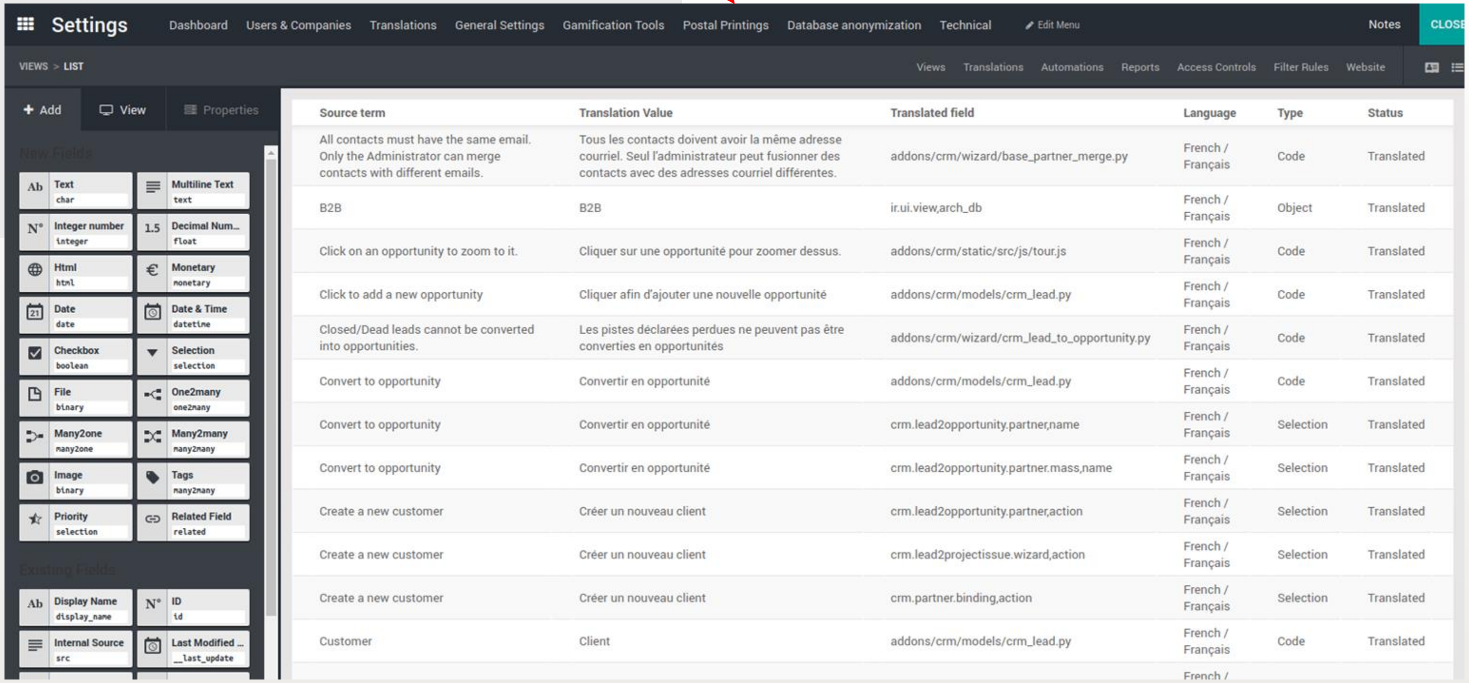
https://www.odoo.com/documentation/10.0/howtos/themes.html
https://www.odoo.com/documentation/10.0/howtos/website.html

ODOO STUDIO

Only in Enterprise Edition

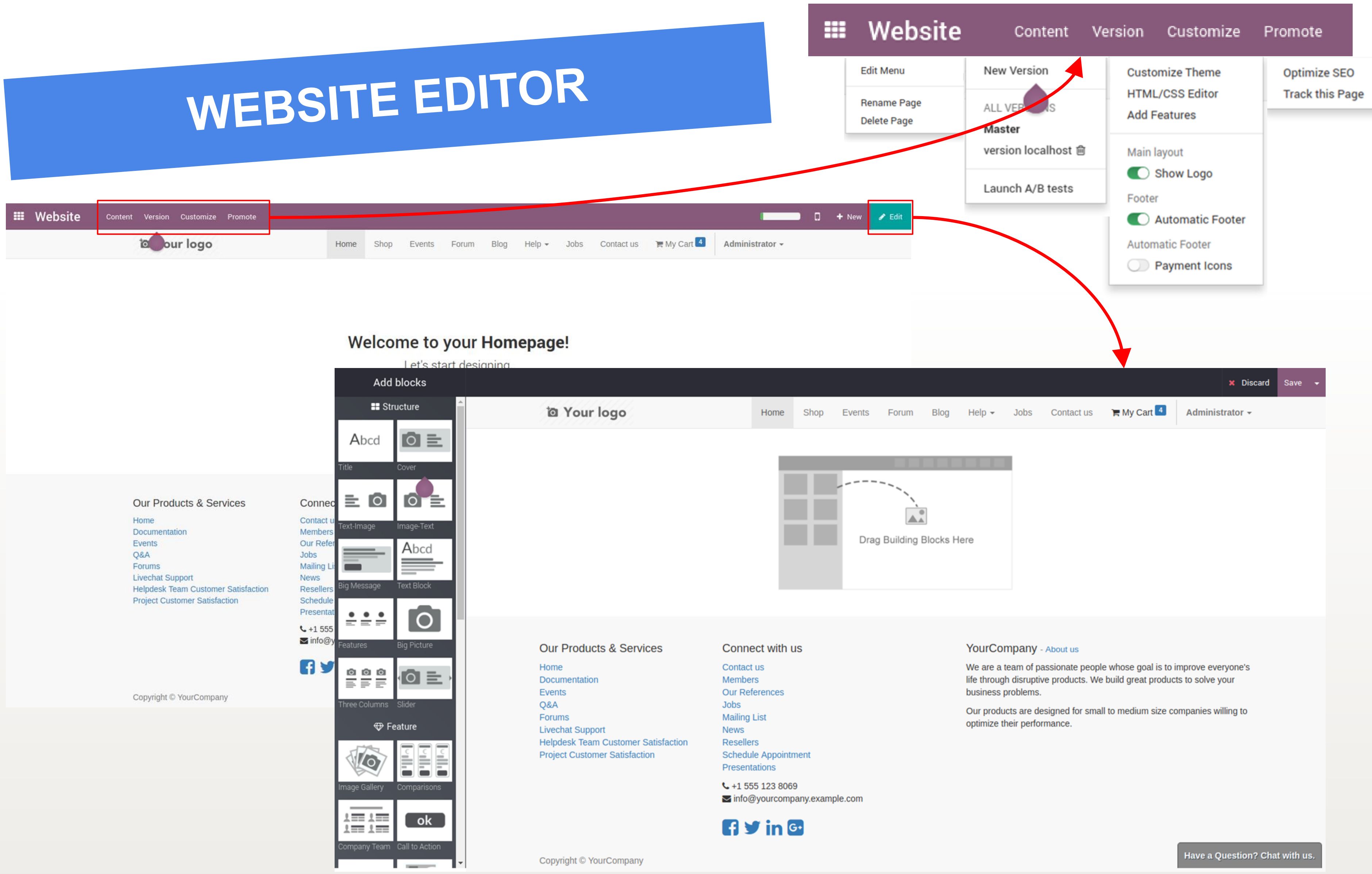






Source term	Translation Value	Translated field	Language	Type	Status
All contacts must have the same email. Only the Administrator can merge contacts with different emails.	Tous les contacts doivent avoir la même adresse courriel. Seul l'administrateur peut fusionner des contacts avec des adresses courriel différentes.	addons/crm/wizard/base_partner_merge.py	French / Français	Code	Translated
B2B	B2B	ir.ui.view,arch_db	French / Français	Object	Translated
Click on an opportunity to zoom to it.	Cliquer sur une opportunité pour zoomer dessus.	addons/crm/static/src/js/tour.js	French / Français	Code	Translated
Click to add a new opportunity	Cliquer afin d'ajouter une nouvelle opportunité	addons/crm/models/crm_lead.py	French / Français	Code	Translated
Closed/Dead leads cannot be converted into opportunities.	Les pistes déclarées perdues ne peuvent pas être converties en opportunités	addons/crm/wizard/crm_lead_to_opportunity.py	French / Français	Code	Translated
Convert to opportunity	Convertir en opportunité	addons/crm/models/crm_lead.py	French / Français	Code	Translated
Convert to opportunity	Convertir en opportunité	crm.lead2opportunity.partner,name	French / Français	Selection	Translated
Convert to opportunity	Convertir en opportunité	crm.lead2opportunity.partner.mass,name	French / Français	Selection	Translated
Create a new customer	Créer un nouveau client	crm.lead2opportunity.partner,action	French / Français	Selection	Translated
Create a new customer	Créer un nouveau client	crm.lead2projectissue.wizard,action	French / Français	Selection	Translated
Create a new customer	Créer un nouveau client	crm.partner.binding,action	French / Français	Selection	Translated
Customer	Client	addons/crm/models/crm_lead.py	French / Français	Code	Translated
			French /		

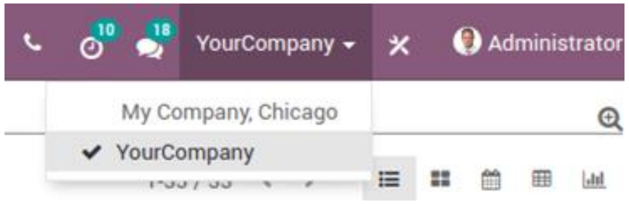
WEBSITE EDITOR



We are a team of passionate people whose goal is to improve everyone's life through disruptive products. We build great products to solve your business problems.

Our products are designed for small to medium size companies willing to optimize their performance.

MULTI-COMPANY



Odoo can manage multiple companies in a same database:

- multi-company = documents sharing
  By default, thanks to record rules, a document linked to a child company is accessible to its parent company but not to its daughters; in other words, only records related to the logged-on user's company (or daughters) are displayed
- inter-company = documents synchronization
  e.g. automatically create a sale / purchase order when one company buys / sells to another one, same for customer and supplier invoices

A company-dependent field is a field whose displayed value depends on the company of the logged-on user.

Extended use cases

- PubAudit: a group composed of several subsidiaries, one chart of accounts by subsidiary
- Casden: a company composed of several establishments, one common chart of accounts for all establishments; each journal entry must be linked to an establishment in order to comply with French law
- In these two cases, users log on parent company

=> Solution: smile_multi_company_* addons allow to use the company carried by the document, instead of logger-on user's company, in order to get company-dependent fields value

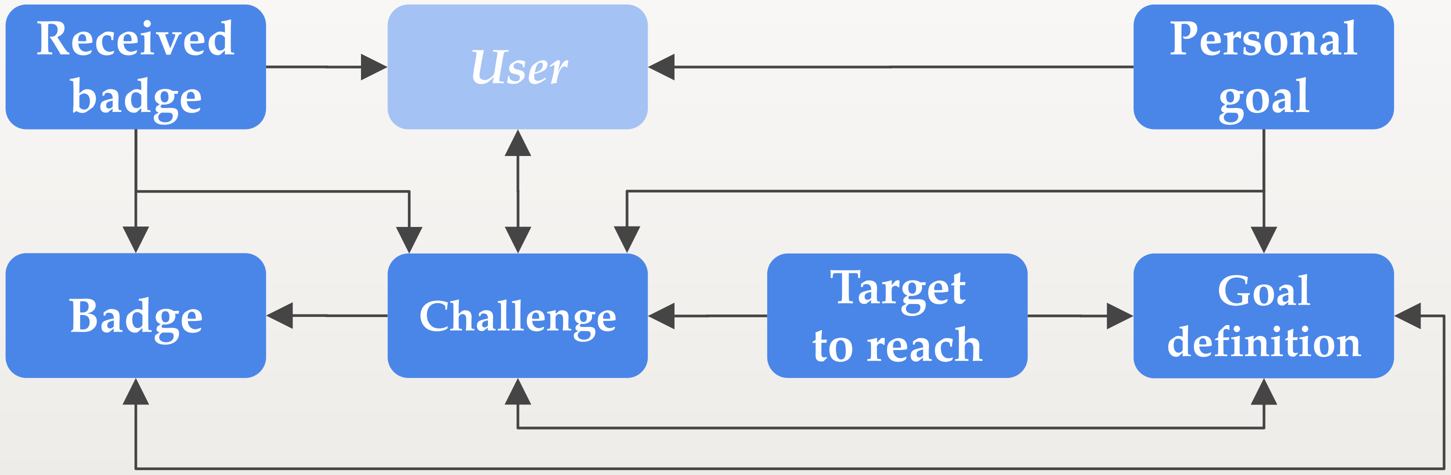
GAMIFICATION

The Gamification module provides ways to evaluate and motivate the users of Odoo, e.g. sales persons.

The users can be evaluated using goals and numerical objectives to reach. Goals are assigned through challenges to evaluate and compare members of a team with each others and through time.

For non-numerical achievements, badges can be granted to users. From a simple "thank you" to an exceptional achievement, a badge is an easy way to express gratitude to a user for their good work.

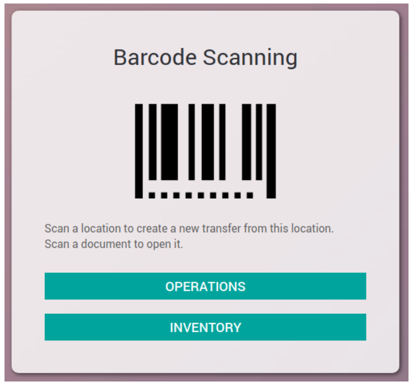
Both goals and badges are flexibles and can be adapted to a large range of modules and actions. When installed, this module creates easy goals to *help new users to discover Odoo and configure their user profile*.



KEYBOARD SHORTCUTS

Functionality	Keyboard shortcut
Create a new record	Alt + c
Save a record	Alt + s
Edit a record (adapt)	Alt + a
Discard a record modification (junk)	Alt + j
Open to list view	Alt + l
Open to kanban view	Alt + k
Open the previous record	Alt + p
Open the next record	Alt + n
Toggle home menu	Alt + h

BARCODES


Barcode Scanning
Scan a location to create a new transfer from this location.
Scan a document to open it.
OPERATIONS
INVENTORY

Odoo provides support for barcode scanning and parsing:

- Scanning

Use a USB scanner (that mimics keyboard inputs) in order to work with barcodes in Odoo.
The scanner must be configured to use no prefix and a carriage return or tab as suffix.
The delay between each character input must be less than or equal to 50 milliseconds.
Most barcode scanners will work out of the box.
However, make sure the scanner uses the same keyboard layout as the device it's plugged in.
Either by setting the device's keyboard layout to US QWERTY (default value for most readers) or by changing the scanner's keyboard layout (check the manual).

- Parsing

The barcodes are interpreted using the rules defined by a nomenclature. It provides the following features:

❏ Patterns to identify barcodes containing a numerical value (e.g. weight, price)
❏ Definition of barcode aliases that allow to identify the same product with different barcodes
❏ Support for encodings EAN-13, EAN-8 and UPC-A

Technically, use the mixin class *barcodes.barcode_events_mixin* for object reacting when a barcode is scanned in a form view which contains `<field name="_barcode_scanned" widget="barcode_handler"/>`.
Models using this mixin must implement the method *on_barcode_scanned(self, barcode)*. It works like an onchange and receives the scanned barcode in parameter.

ODOO TOOLS

from odoo import tools

- tools.**file_open**(name, mode="r", subdir='addons', pathinfo=False)
  Open a file from the OpenERP root, using a subdir folder
- tools.**float_compare**(value1, value2, precision_digits=None, precision_rounding=None)
  Compare ``value1`` and ``value2`` after rounding them according to the given precision. A value is considered lower/greater than another value if their rounded value is different. This is not the same as having a non-zero difference!
- tools.**safe_eval**(expr, globals_dict=None, locals_dict=None, mode="eval", nocopy=False, locals_builtins=False)
  Evaluates a string that contains an expression that mostly uses Python constants, arithmetic expressions and the objects directly provided in context. This can be used to e.g. evaluate an Odoo domain expression from an untrusted source
- tools.**ustr**(value, hint_encoding='utf-8', errors='strict')
  Similar to the builtin `unicode`, except that it may try multiple encodings to find one that works for decoding `value`, and defaults to 'utf-8' first

ODOO DATE UTILS

from odoo import fields

- fields.Date.**today**()
- fields.Datetime.**now**()
- fields.Date*.**from_string**(date)
- fields.Date*.**to_string**(date)
- fields.Date.**context_today**(record, timestamp=None)
  => return timestamp (in fact, datetime.date) or today in the timezone passed in the record's context or in the user's timezone
- fields.Datetime.**context_timestamp**(record, timestamp)
  => return timestamp (in fact, datetime.date) in the timezone passed in the record's context or in the user's timezone. Don't use it for default value.

Note: dates are stored in database in UTC and converted in JS before being displayed according to the user timezone

ASYNCHRONOUS

The *best* way to speed up an user action is to run it:

- later through a scheduled action
- in the background in a job queue *or a new thread with a new environment and a new cursor*

queue_job by OCA

```
from odoo import api, models
from odoo.addons.queue_job.job import job


class MyModel(models.Model):
   _name = 'my.model'

   @api.multi
   @job
   def my_method(self, a, k=None):
       ...


class MyOtherModel(models.Model):
    _name = 'my.other.model'

    @api.multi
    def button_do_stuff(self):
        self.env['my.model'].with_delay().my_method('a', k=2)
```

CODING GUIDELINES

Based on the Odoo Guidelines with adaptations to improve their guidelines and make them more suitable for our projects own needs.

This page introduces the coding guidelines for projects developed by Smile.These guidelines aim to improve the quality of the code: better readability of source, better maintainability, better stability and fewer regressions.

Differences with Odoo Guidelines

- Module structure
  - Separating data and demo data xml folders
- XML files
  - Not changing xml_ids while inheriting
- PEP8 Options
  - Fuller PEP8 compliance. The only exceptions:
    - E501: line too long (< 120 characters is good)
    - F401: module imported but unused, only in __init__.py
- Imports
  - Using relative import for local files
- Idioms
  - Use meaningful variable/class/method names rather than docstring
  - Use massively API decorators to avoid poor performance
- Symbols
  - The compute method pattern is _get_field_name
  - The inverse method pattern is _set_field_name

BEST PRACTICES

Performance

- Computed fields : store them if
  - complex to compute or recursively defined
  - used in a large export
  - displayed on list/kanban/graph/gantt/calendar views
- Many2one relations
  - Index field with a large cardinality (*index=True*)
  - Add *auto_join=True* for "parent" fields - Warning: run faster searches by using SQL joins, but bypass security rules

    ```
    order_id = fields.Many2one('sale.order', 'Sale Order',... auto_join=True)
    ```

    You can use it on One2many fields too
- Hierarchical relations
  - Use *_parent_store=True* and define *parent_left* and *parent_right* fields
- Methods
  - Browse all records before accessing their fields

    ```
    for record in self.browse(ids):
         if record.partner_id ...:  # the loop issues O(1) queries
    ```

  - Prefer search with *auto_join* rather than mapped with a large dot-separated sequence of field names

BEST PRACTICES

Security

- Use sandboxed Python expression evaluation

```
from odoo.tools.safe_eval import safe_eval
```

- To avoid SQL injections, use *cr.execute(query, params)* rather than *cr.execute(query % params)*
- To avoid XSS (cross-site scripting) vulnerability, *t-raw* must not be used on any data which may contain non-escaped user-provided content

Maintainability

- Models
  - *on_change* doesn't encapsulate business logic. It's just an input help
  - Use *AbstractModel* to encapsulate business logic shared between different models

```
class FactoryOrder(models.AbstractModel):
    _name = 'factory.order'
    _description = 'Factory Order'

class ProductionOrder(models.Model):
     _name = 'mrp.production'
     _inherit = ['mrp.production', 'factory.order']

class PurchaseOrder(models.Model):
     _name = 'purchase.order'
     _inherit = ['purchase.order', 'factory.order']
```

BEST PRACTICES

- Fields
  - Default functions should be declared with a lambda call on *self* in order to allow this overriding

```
a_field(..., default=lambda self: self._default_get())
```

  - If a *Many2one* field is required, specify *ondelete=cascade* / *restrict* (*set null* by default)
  - For all computed fields, *@depends* is required in order to work as *@onchange* in UI
- Methods
  - Prefer *BaseModel._patch_method* to monkey patch
  - Add hooks

```
def _get_vals(self):
    return {...}

def my_method(self):
    vals = self._get_vals()  # easy to override
    self.write(vals)
```

  - Use skilfully API decorators

```
@api.one
def name_get(self):
    return self.id, self.name
self.name_get()  # Returns [(id1, name1), (id2, name2)...]
```

BEST PRACTICES

- Views inheritance
  - Use *mode=primary* rather than add too many *attrs={'invisible': ...}* if this view is accessible from a specific menu. That creates a copy of the inherited view, i.e. that doesn't modify the inherited view
- Files storage
  - No files in binary fields, use the filestore. Faster dumps and easier to rsync backups

PERFS ANALYSIS

Many factors can impact performance

- Harware bottlenecks
  - Check Hosting *munin* graphs
- Business logic burning CPU
  - Use *odoo.tools.profile* method decorator
  - Activate logs *(see the next slide)*
- Transaction locking in the database
  - See *https://wiki.postgresql.org/wiki/Lock_Monitoring*
- SQL query performance
  - Use *pg_badger* to analyse the query logs

PERFS ANALYSIS

Logging configuration *./odoo-bin .. --log-**	**Description**
--log-request	Log RPC requests `odoo.http.rpc.request: search_read: sale.order None: time:0.048s mem: 663472k -> 663472k (diff: 0k)`
--log-response	Log RPC responses `odoo.http.rpc.response: search_read: sale.order None: time:0.071s mem: 663460k -> 663460k (diff: 0k), {'length': 11, 'records': [{u'amount_total': 675000.0, u'currency_id': (1, u'EUR'), u'date_order': '2017-08-23 09:24:24', 'id': 11, u'message_needaction': False, u'name': u'SO010', u'partner_id': (48, u'Optique Daniel'), u'state': u'sent', u'user_id': (1, u'Administrator')},]`

--log-web	Log HTTP requests
	`odoo.http: HTTP sessions stored in: ~/.local/share/Odoo/sessions`
	`odoo.http: Loading website_event`

--log-sql	Log and check SQL queries sent to the database
	With that, we found cases where cache prefetching was not working as expected
	`odoo.sql_db: query: SELECT 1 FROM ir_module_module WHERE name=%s AND latest_version=%s`
	`odoo.sql_db: table: ir_module_module: 0:00:00.000814/1`

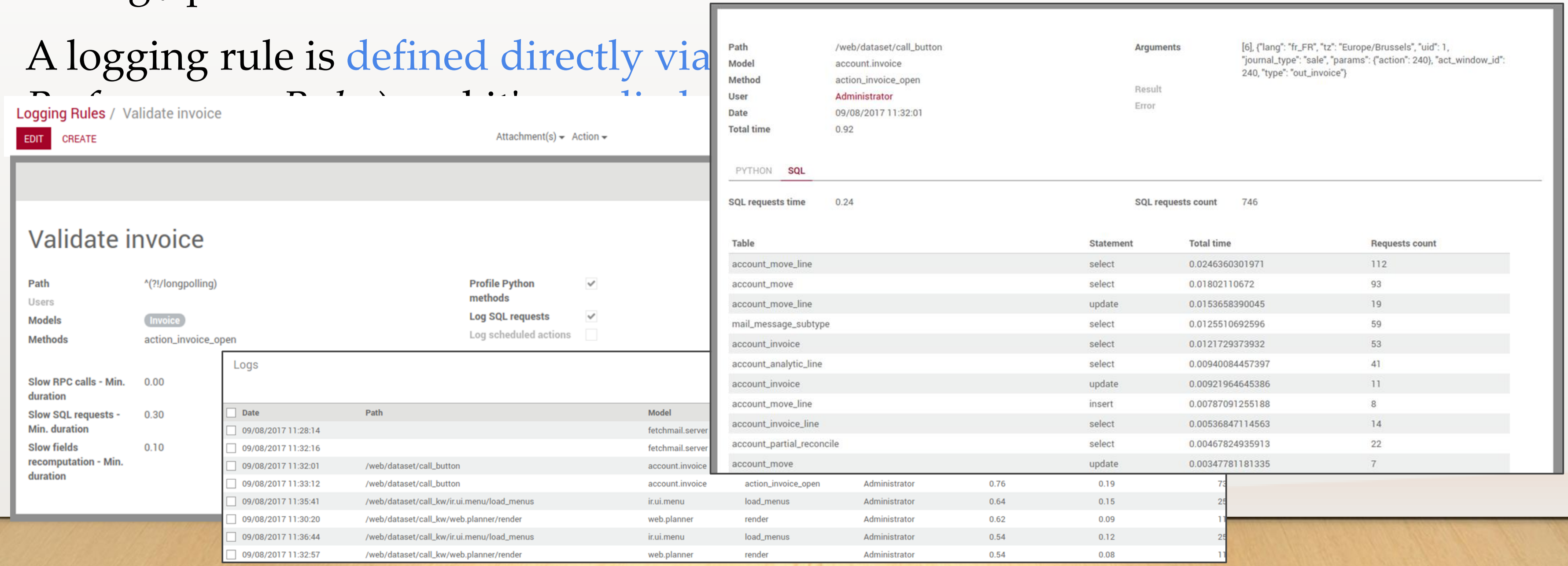
PERFS ANALYSIS

Smile developed an addon *odoo_perf_analyzer*, available from Odoo v8.0, to log in function of rules:

- each RPC requests linked to a model
- Python method profiling
- SQL queries stats

That allows to discover the longest user actions in cumulative, in a customer testing / production environment.

A logging rule is defined directly via



Validate invoice

Path	^(?!/longpolling)	Profile Python methods	✓
Users		Log SQL requests	✓
Models	Invoice	Log scheduled actions	
Methods	action_invoice_open		
Slow RPC calls - Min. duration	0.00		
Slow SQL requests - Min. duration	0.30		
Slow fields recomputation - Min. duration	0.10		

Logs

Date	Path	Model					
09/08/2017 11:28:14		fetchmail.server					
09/08/2017 11:32:16		fetchmail.server					
09/08/2017 11:32:01	/web/dataset/call_button	account.invoice					
09/08/2017 11:33:12	/web/dataset/call_button	account.invoice	action_invoice_open	Administrator	0.76	0.19	73
09/08/2017 11:35:41	/web/dataset/call_kw/ir.ui.menu/load_menus	ir.ui.menu	load_menus	Administrator	0.64	0.15	25
09/08/2017 11:30:20	/web/dataset/call_kw/web.planner/render	web.planner	render	Administrator	0.62	0.09	11
09/08/2017 11:36:44	/web/dataset/call_kw/ir.ui.menu/load_menus	ir.ui.menu	load_menus	Administrator	0.54	0.12	25
09/08/2017 11:32:57	/web/dataset/call_kw/web.planner/render	web.planner	render	Administrator	0.54	0.08	11

Path	/web/dataset/call_button	Arguments	[6], {"lang": "fr_FR", "tz": "Europe/Brussels", "uid": 1, "journal_type": "sale", "params": {"action": 240}, "act_window_id": 240, "type": "out_invoice"}
Model	account.invoice		
Method	action_invoice_open	Result	
User	Administrator	Error	
Date	09/08/2017 11:32:01		
Total time	0.92		

PYTHON SQL

SQL requests time	0.24	SQL requests count	746

Table	Statement	Total time	Requests count
account_move_line	select	0.0246360301971	112
account_move	select	0.01802110672	93
account_move_line	update	0.0153658390045	19
mail_message_subtype	select	0.0125510692596	59
account_invoice	select	0.0121729373932	53
account_analytic_line	select	0.00940084457397	41
account_invoice	update	0.00921964645386	11
account_move_line	insert	0.00787091255188	8
account_invoice_line	select	0.00536847114563	14
account_partial_reconcile	select	0.00467824935913	22
account_move	update	0.00347781181335	7