odoo

First steps in Odoo dev

Richard Mathot - rim@odoo.com

Agenda

For beginners:

- Architecture of Odoo
- Module Open Academy

odoo

Architecture of Odoo

Architecture of Odoo

- Three-tier client/server/database
- Web client in Javascript
- Server and backend modules in Python
  - MVC framework

odoo

Module Open Academy

The Module

- Manage courses, sessions, and subscriptions
- Learn
  - Structure of a module
  - Definition of data models
  - Definition of views and menus

Structure of a Module

An Odoo module is

- a Python(2) module (data models), with
- a manifest file,
- XML and CSV data files (base data, views, menus),
- frontend resources (Javascript, CSS).

The Open Academy Module

The manifest file __openerp__.py:

```
{
    'name': 'Open Academy',
    'version': '1.0',
    'category': 'Tools',
    'summary': 'Courses, Sessions, Subscriptions',
    'description': "...",
    'depends': ['base'],
    'data': ['views/views.xml'],
    'demo': [],
    'application': True,
}
```

The Course Model

A model and its fields are defined in a Python class:

```
from openerp import fields, models


class Course(models.Model):
    _name = 'openacademy.course'

    name = fields.Char(string='Title', required=True)
    description = fields.Text()
```

The Menu as XML data

```xml
<?xml version="1.0" encoding="UTF-8"?>
<odoo>

    <menuitem name="Open Academy" id="openacademy_root"/>

    <menuitem name="General" id="openacademy_general" parent="openacademy_root"/>

    <record model="ir.actions.act_window" id="action_courses">
        <field name="name">Courses</field>
        <field name="res_model">openacademy.course</field>
        <field name="view_mode">tree,form</field>
    </record>

    <menuitem name="Courses" id="menu_courses" parent="openacademy_general"
              sequence="1" action="action_courses"/>

</odoo>
```

Let's add a Form View

```
<record model="ir.ui.view" id="course_form">
    <field name="name">Course Form View</field>
    <field name="model">openacademy.course</field>
    <field name="arch" type="xml">
        <form string="Course" version="7.0">
            <sheet>
                <h1>
                    <field name="name" placeholder="Course Title"/>
                </h1>
                <notebook>
                    <page string="Description">
                        <field name="description"/>
                    </page>
                </notebook>
            </sheet>
        </form>
    </field>
</record>
```

The Session Model

```
from openerp import fields, models


class Session(models.Model):
    _name = 'openacademy.session'

    name = fields.Char(required=True)
    start_date = fields.Date()
    duration = fields.Integer(help="Duration in days")
    seats = fields.Integer(string="Number of Seats")
```

Relational Fields

Let us link sessions to courses and instructors:

```
class Session(models.Model):
    _name = 'openacademy.session'

    ...

course = fields.Many2one('openacademy.course', required=True)
instructor = fields.Many2one('res.partner')
```

Relational Fields

Let us back-link courses and sessions:

```
class Course(models.Model):
    _name = 'openacademy.course'

    ...

    responsible = fields.Many2one('res.users')
    sessions = fields.One2many('openacademy.session', 'course')
```

Relational Fields

Let us link sessions to partners for attendee subscription:

```
class Session(models.Model):
    _name = 'openacademy.session'

    ...

    attendees = fields.Many2many('res.partner')
```

Computed Fields

The value of those fields is computed:

```
class Session(models.Model):
    _name = 'openacademy.session'

    ...

    taken_seats = fields.Float(compute='_compute_taken_seats',
                               string="Percentage of taken seats")

    @api.ones
    @api.depends('attendees', 'seats')
    def _compute_taken_seats(self):
        if self.seats:
            self.taken_seats = 100.0 * len(self.attendees) / self.seats
        else:
            self.taken_seats = 0.0
```

About self

Model instances are recordsets.

A recordset is an hybrid concept:

- collection of records
- record

```
for session in self:
    print session.name
    print session.course.name

assert self.name == self[0].name
```

Feedback with "Onchange" Methods

Modify form values when some field is filled in:

```
class Session(models.Model):
    _name = 'openacademy.session'

    ...

    @api.onchange('course')
    def _onchange_course(self):
        if not self.name:
            self.name = self.course.name
```

Default Values

Specify the initial value to use in a form:

```
class Session(models.Model):
    _name = 'openacademy.session'

    ...

    # Special field for archiving
    active = fields.Boolean(default=True)
    start_date = fields.Date(default=fields.Date.today())

    ...
```

Model Constraints

Prevent bad data:

```
from openerp.exceptions import UserError

class Session(models.Model):
    _name = 'openacademy.session'

    ...

    @api.one
    @api.constrains('instructor', 'attendees')
    def _check_instructor(self):
        if self.instructor in self.attendees:
            raise UserError("Instructor of session '%s' "
                "cannot attend its own session" % self.name)
```

More Stuff

- Extend existing models
- Many view types
- Workflows
- Reports
- Security
- Translations

odoo

First steps in Odoo dev

Conclusion

- Modules have a simple structure
- Model definition intuitive and efficient
  - uses Python standards (decorators, descriptors)
  - recordsets provide support for "batch" processing
  - many model hooks (default values, constraints, computed fields)

Conclusion

- https://www.odoo.com/documentation/
  - Tutorials
  - Framework reference

odoo

First steps in Odoo dev

Richard Mathot - rim@odoo.com