Odoo ERP Development Guide

1. Core Architecture Overview

Odoo follows an **MVC-like architecture** with these key components:

- **Models** (Python) - Business logic and database structure
- **Views** (XML) - User interface definitions
- **Controllers** (Python) - Handle HTTP requests/responses
- **Data** (XML/CSV) - Initial data and demo data

2. Module Structure

Every Odoo module follows this standard structure:

```
my_module/
├── __init__.py             # Import models and controllers
├── __manifest__.py         # Module metadata and dependencies
├── models/
│   ├── __init__.py
│   └── my_model.py         # Business logic
├── views/
│   ├── my_model_views.xml  # Form, tree, search views
│   └── menu.xml            # Menu items
├── security/
│   ├── ir.model.access.csv # Access rights
│   └── security.xml        # Record rules
├── data/
│   └── data.xml            # Default data
├── demo/
│   └── demo.xml            # Demo data
├── controllers/
│   ├── __init__.py
│   └── main.py             # Web controllers
└── static/
    ├── description/
    │   └── icon.png
    └── src/
        ├── js/
        ├── css/
        └── xml/
```

3. Essential Files

manifest.py

python

```python
{
    'name': 'My Module',
    'version': '17.0.1.0.0',
    'category': 'Sales',
    'summary': 'Brief description',
    'depends': ['base', 'sale'],  # Module dependencies
    'data': [
        'security/ir.model.access.csv',
        'views/my_model_views.xml',
        'data/data.xml',
    ],
    'demo': ['demo/demo.xml'],
    'installable': True,
    'application': False,
    'auto_install': False,
}
```

init.py (root)

python

```python
from . import models
from . import controllers
```

4. Models - The Heart of Odoo

Basic Model Structure

python

```python
from odoo import models, fields, api
from odoo.exceptions import ValidationError

class MyModel(models.Model):
    _name = 'my.model'
    _description = 'My Model Description'
    _inherit = ['mail.thread', 'mail.activity.mixin']  # Optional inheritance
    _order = 'name desc'

    # Fields
    name = fields.Char(string='Name', required=True, index=True)
    description = fields.Text(string='Description')
    active = fields.Boolean(default=True)
    state = fields.Selection([
        ('draft', 'Draft'),
        ('confirmed', 'Confirmed'),
        ('done', 'Done')
    ], default='draft', tracking=True)

    # Relational fields
    partner_id = fields.Many2one('res.partner', string='Customer', required=True)
    line_ids = fields.One2many('my.model.line', 'parent_id', string='Lines')
    tag_ids = fields.Many2many('my.tags', string='Tags')

    # Computed fields
    total_amount = fields.Float(compute='_compute_total', store=True)

    # Constraints
    _sql_constraints = [
        ('name_unique', 'UNIQUE(name)', 'Name must be unique!')
    ]

    @api.depends('line_ids.amount')
    def _compute_total(self):
        for record in self:
            record.total_amount = sum(record.line_ids.mapped('amount'))

    @api.constrains('name')
    def _check_name(self):
        for record in self:
            if len(record.name) < 3:
                raise ValidationError("Name must be at least 3 characters!")
```

```
    def action_confirm(self):
        self.write({'state': 'confirmed'})
        return True
```

Model Types

- **models.Model** - Regular persistent model (database table)
- **models.TransientModel** - Temporary wizard model (auto-deleted)
- **models.AbstractModel** - Abstract model for inheritance only

5. Inheritance Patterns

Classical Inheritance (_inherit with same _name)

python

```
class SaleOrder(models.Model):
    _inherit = 'sale.order'

    custom_field = fields.Char('Custom Field')

    def action_confirm(self):
        # Add custom logic
        res = super().action_confirm()
        # More custom logic
        return res
```

Extension Inheritance (_inherit with different _name)

python

```
class ProductTemplate(models.Model):
    _name = 'product.template'
    _inherit = ['product.template', 'website.published.mixin']
```

Delegation Inheritance (_inherits)

python

```python
class User(models.Model):
    _name = 'res.users'
    _inherits = {'res.partner': 'partner_id'}

    partner_id = fields.Many2one('res.partner', required=True)
```

6. Common Field Types

python

```python
# Basic fields
name = fields.Char(string='Name', size=128)
description = fields.Text('Description')
amount = fields.Float('Amount', digits=(16, 2))
quantity = fields.Integer('Quantity')
is_active = fields.Boolean('Active', default=True)
date = fields.Date('Date')
datetime = fields.Datetime('DateTime', default=fields.Datetime.now)

# Selection
state = fields.Selection([
    ('draft', 'Draft'),
    ('done', 'Done')
], string='Status')

# Relational
partner_id = fields.Many2one('res.partner', 'Customer')
line_ids = fields.One2many('sale.order.line', 'order_id', 'Lines')
tag_ids = fields.Many2many('product.tag', string='Tags')

# Computed
total = fields.Float(compute='_compute_total', store=True)

# Related (shortcut to related field)
partner_country_id = fields.Many2one('res.country',
    related='partner_id.country_id', readonly=True)
```

7. Views (XML)

Form View

xml

```
<record id="view_my_model_form" model="ir.ui.view">
  <field name="name">my.model.form</field>
  <field name="model">my.model</field>
  <field name="arch" type="xml">
    <form>
      <header>
        <button name="action_confirm" string="Confirm"
          type="object" class="btn-primary"/>
        <field name="state" widget="statusbar"/>
      </header>
      <sheet>
        <group>
          <group>
            <field name="name"/>
            <field name="partner_id"/>
          </group>
          <group>
            <field name="date"/>
            <field name="total_amount"/>
          </group>
        </group>
        <notebook>
          <page string="Lines">
            <field name="line_ids">
              <tree editable="bottom">
                <field name="product_id"/>
                <field name="quantity"/>
                <field name="price"/>
              </tree>
            </field>
          </page>
        </notebook>
      </sheet>
      <div class="oe_chatter">
        <field name="message_follower_ids"/>
        <field name="message_ids"/>
      </div>
    </form>
  </field>
</record>
```

Tree View

xml

```xml
<record id="view_my_model_tree" model="ir.ui.view">
  <field name="name">my.model.tree</field>
  <field name="model">my.model</field>
  <field name="arch" type="xml">
    <tree decoration-info="state=='draft'"
          decoration-success="state=='done'">
      <field name="name"/>
      <field name="partner_id"/>
      <field name="date"/>
      <field name="total_amount"/>
      <field name="state"/>
    </tree>
  </field>
</record>
```

Search View

xml

```xml
<record id="view_my_model_search" model="ir.ui.view">
  <field name="name">my.model.search</field>
  <field name="model">my.model</field>
  <field name="arch" type="xml">
    <search>
      <field name="name"/>
      <field name="partner_id"/>
      <filter name="draft" string="Draft"
        domain="[('state','=','draft')]"/>
      <group expand="0" string="Group By">
        <filter name="group_partner" string="Customer"
          context="{'group_by':'partner_id'}"/>
      </group>
    </search>
  </field>
</record>
```

8. Security (ir.model.access.csv)

csv

```
id,name,model_id:id,group_id:id,perm_read,perm_write,perm_create,perm_unlink
access_my_model_user,my.model.user,model_my_model,base.group_user,1,1,1,0
access_my_model_manager,my.model.manager,model_my_model,base.group_system,1,1,1,1
```

9. Common ORM Methods

python

```python
# Create
record = self.env['my.model'].create({
    'name': 'Test',
    'partner_id': partner.id
})

# Search
records = self.env['my.model'].search([
    ('state', '=', 'draft'),
    ('date', '>=', '2024-01-01')
], limit=10, order='date desc')

# Browse (get by ID)
record = self.env['my.model'].browse(record_id)

# Write
record.write({'state': 'confirmed'})
# or
record.state = 'confirmed'

# Delete
record.unlink()

# Exists check
if record.exists():
    print("Record exists")

# Mapped (get field values)
names = records.mapped('name')
partner_ids = records.mapped('partner_id.id')

# Filtered
draft_records = records.filtered(lambda r: r.state == 'draft')

# Sorted
sorted_records = records.sorted(key=lambda r: r.date)
```

10. Controllers (Web Routes)

```
python
```

```python
from odoo import http
from odoo.http import request

class MyController(http.Controller):

    @http.route('/my/page', type='http', auth='user', website=True)
    def my_page(self, **kwargs):
        records = request.env['my.model'].search([])
        return request.render('my_module.my_template', {
            'records': records
        })

    @http.route('/api/data', type='json', auth='user')
    def get_data(self, **kwargs):
        data = request.env['my.model'].search_read([], ['name', 'state'])
        return {'data': data}
```

11. Best Practices

1. **Use proper naming conventions**
   - Models: `snake_case` (e.g., `sale.order`)
   - Classes: `PascalCase` (e.g., `SaleOrder`)
   - Methods: `snake_case` (e.g., `action_confirm`)
2. **Always use self.env['model.name']** instead of pooler
3. **Use @api decorators properly**
   - `@api.depends` for computed fields
   - `@api.constrains` for validations
   - `@api.onchange` for form interactions
4. **Batch operations** - Work with recordsets, not loops when possible
5. **Security** - Always define access rights and record rules
6. **Translations** - Use `_()` for translatable strings
7. **Performance** - Use `store=True` wisely and add indexes where needed

12. Debugging Tips

```python
```

```
# Logging
import logging
_logger = logging.getLogger(__name__)

_logger.info('Info message')
_logger.warning('Warning message')
_logger.error('Error message')

# Debug in methods
import pdb; pdb.set_trace()

# Print SQL queries
self.env.cr.mogrify(query, params)
```

13. Useful Commands

bash

```
# Upgrade module
odoo-bin -u my_module -d database_name

# Install module
odoo-bin -i my_module -d database_name

# Update module list
# Settings > Apps > Update Apps List

# Check logs
tail -f /var/log/odoo/odoo.log
```

This guide covers the essentials. For deeper customization, explore the official Odoo documentation and study existing modules in your Odoo installation!