

Complete Odoo 18.0 Development Guide

Introduction

This comprehensive guide covers the entire Odoo 18.0 development lifecycle, from initial setup to production deployment. It provides practical examples, code snippets, and real-world scenarios for developers at all skill levels who want to build robust Odoo applications.

1. Development Environment Setup and Requirements

System requirements for Odoo 18.0

Operating Systems:

- **Linux (Recommended):** Ubuntu 22.04 LTS or 24.04, Debian 12 (Bookworm) or above [Odoo](#)
- **Windows:** Supported but Linux strongly recommended for production
- **macOS:** Supported for development

Core Dependencies:

- **Python:** 3.10 or later (Python 3.7-3.9 no longer supported) [Odoo](#)
- **PostgreSQL:** 12.0 or above (13+ recommended) [Odoo](#)
- **Additional:** wkhtmltopdf 0.12.6, [Odoo](#) Node.js for RTL support

Installation methods guide

Source Code Installation (Recommended for Developers)

```
bash
```

```
# Clone repositories
```

```
git clone https://github.com/odoo/odoo.git --depth 1 --branch 18.0 --single-branch  
cd odoo
```

```
# Create virtual environment
```

```
python3 -m venv venv  
source venv/bin/activate
```

```
# Install dependencies
```

```
pip3 install -r requirements.txt
```

```
# Install system dependencies (Ubuntu/Debian)
```

```
sudo apt install python3-pip python3-dev libxml2-dev libxslt1-dev libldap2-dev \  
    libsasl2-dev libssl-dev libpq-dev libjpeg-dev build-essential
```

```
# Setup PostgreSQL
```

```
sudo -u postgres createuser -s $USER  
createdb odoo18-dev
```

```
# Run Odoo in development mode
```

```
./odoo-bin --addons-path=addons --database=odoo18-dev --dev=reload,qweb,werkzeug,xml
```

Docker Installation

yaml

```
# docker-compose.yml
version: '3'
services:
  odoo:
    image: odoo:18.0
    depends_on:
      - postgres
    ports:
      - "127.0.0.1:8069:8069"
    volumes:
      - data:/var/lib/odoo
      - ./config:/etc/odoo
      - ./addons:/mnt/extra-addons
    environment:
      - HOST=postgres
      - USER=odoo
      - PASSWORD=odoo
  postgres:
    image: postgres:13
    environment:
      - POSTGRES_DB=postgres
      - POSTGRES_USER=odoo
      - POSTGRES_PASSWORD=odoo
    volumes:
      - db:/var/lib/postgresql/data
volumes:
  data:
  db:
```

IDE Configuration

VSCode Setup:

json

```
// .vscode/settings.json
{
  "python.defaultInterpreterPath": "/path/to/venv/bin/python",
  "python.linting.enabled": true,
  "python.linting.flake8Enabled": true,
  "files.associations": {
    "*.xml": "xml"
  }
}
```

2. Odoo Framework Architecture and Structure

Understanding the MVC architecture

Odoo follows a three-tier MVC architecture: [Maasmind -](#)

- **Model (Data Tier):** PostgreSQL database with automatic table generation [Technaureus](#)
- **View (Presentation Tier):** XML files, HTML5, JavaScript (OWL framework), CSS
- **Controller (Logic Tier):** Python classes handling business logic and HTTP requests [Odoo](#) [Odoo](#)

Core components overview

ORM (Object-Relational Mapping):

```
python

from odoo import models, fields, api

class EstateProperty(models.Model):
    _name = "estate.property"
    _description = "Real Estate Property"

    name = fields.Char(required=True)
    expected_price = fields.Float()

    @api.depends('bedrooms')
    def _compute_total_area(self):
        for record in self:
            record.total_area = record.bedrooms * 10
```

Request Lifecycle:

1. HTTP Request arrives
2. Routing determines controller method
3. Authentication and session management
4. Controller processes request
5. Model executes business logic
6. View renders response
7. Response sent to client

3. Module Development Fundamentals

Creating custom modules structure

Module Scaffolding:

bash

```
odoo-bin scaffold estate_management /path/to/addons/
```

Standard Module Structure:

```
estate_management/
├── __init__.py
├── __manifest__.py
├── controllers/
│   ├── __init__.py
│   └── main.py
├── data/
│   └── master_data.xml
├── demo/
│   └── demo_data.xml
├── models/
│   ├── __init__.py
│   └── estate_property.py
├── security/
│   ├── ir.model.access.csv
│   └── security_groups.xml
├── static/
│   ├── description/
│   │   └── icon.png
│   └── src/
│       ├── css/
│       ├── js/
│       └── xml/
├── views/
│   └── estate_property_views.xml
└── wizard/
    └── property_wizard.py
```

Odoo Odoo

Manifest file configuration


```

# __manifest__.py
{
    # Required Keys
    'name': 'Estate Management',
    'version': '18.0.1.0.0',
    'depends': ['base', 'sale', 'account'],

    # Descriptive Keys
    'author': "Your Company",
    'website': "https://www.yourcompany.com",
    'category': 'Real Estate',
    'summary': 'Manage real estate properties and sales',
    'description': ""
Estate Management Module
=====
This module allows you to:
* Manage properties
* Track property sales
* Handle property types
    """
    'license': 'LGPL-3',

    # Data Files
    'data': [
        'security/security_groups.xml',
        'security/ir.model.access.csv',
        'views/estate_property_views.xml',
        'views/estate_menus.xml',
        'wizard/property_wizard_views.xml',
    ],
    'demo': [
        'demo/demo_properties.xml',
    ],

    # Application Configuration
    'application': True,
    'installable': True,

    # External Dependencies
    'external_dependencies': {
        'python': ['requests', 'lxml'],
        'bin': ['wkhtmltopdf']
    },

    # Asset Management (Odoo 18.0)
    'assets': {

```

```
        'web.assets_backend': [  
            'estate_management/static/src/css/estate.css',  
            'estate_management/static/src/js/estate_widget.js',  
        ],  
    },  
}
```

Odoo Odoo

4. Model Development (ORM)

Field types and attributes

python

```
class EstateProperty(models.Model):
    _name = 'estate.property'
    _description = 'Real Estate Property'

    # Basic Fields
    name = fields.Char(string='Property Name', required=True, size=100)
    description = fields.Text(translate=True)
    expected_price = fields.Float(digits=(12, 2), help="Expected selling price")
    bedrooms = fields.Integer(default=1)
    garden = fields.Boolean(default=False)
    date_availability = fields.Date(default=fields.Date.today)

    # Selection Field
    state = fields.Selection([
        ('new', 'New'),
        ('offer_received', 'Offer Received'),
        ('offer_accepted', 'Offer Accepted'),
        ('sold', 'Sold'),
        ('canceled', 'Canceled')
    ], string='Status', default='new', required=True)

    # Relational Fields
    property_type_id = fields.Many2one('estate.property.type', required=True)
    buyer_id = fields.Many2one('res.partner', copy=False)
    offer_ids = fields.One2many('estate.property.offer', 'property_id', string='Offers')
    tag_ids = fields.Many2many('estate.property.tag', string='Tags')

    # Computed Fields
    total_area = fields.Integer(compute='_compute_total_area', store=True)
    best_offer = fields.Float(compute='_compute_best_offer')

    @api.depends('living_area', 'garden_area')
    def _compute_total_area(self):
        for record in self:
            record.total_area = record.living_area + record.garden_area

    @api.depends('offer_ids.price')
    def _compute_best_offer(self):
        for record in self:
            if record.offer_ids:
                record.best_offer = max(record.offer_ids.mapped('price'))
            else:
                record.best_offer = 0.0
```

Model inheritance patterns

python

Classical Inheritance (New Model)

```
class CommercialProperty(models.Model):
```

```
    _name = 'commercial.property'
```

```
    _inherit = 'property.base'
```

```
    _description = 'Commercial Property'
```

```
    business_type = fields.Char()
```

```
    lease_term = fields.Integer()
```

Extension Inheritance (Same Model)

```
class PartnerExtension(models.Model):
```

```
    _inherit = 'res.partner'
```

```
    property_ids = fields.One2many('estate.property', 'buyer_id', string='Properties')
```

Delegation Inheritance

```
class PropertyWithDocument(models.Model):
```

```
    _name = 'property.with.document'
```

```
    _inherits = {'property.document': 'document_id'}
```

```
    document_id = fields.Many2one('property.document', required=True, ondelete='cascade')
```

Odoo Odoo

Constraints and validations

python

```
class EstateProperty(models.Model):
    _name = 'estate.property'

    # SQL Constraints
    _sql_constraints = [
        ('check_expected_price', 'CHECK(expected_price > 0)',
         'Expected price must be strictly positive'),
        ('unique_property_name', 'UNIQUE(name)',
         'Property name must be unique'),
    ]

    # Python Constraints
    @api.constrains('expected_price', 'selling_price')
    def _check_selling_price(self):
        for record in self:
            if record.selling_price and record.expected_price:
                min_price = record.expected_price * 0.9
                if float_compare(record.selling_price, min_price, precision_digits=2) < 0:
                    raise ValidationError(
                        f"Selling price cannot be lower than 90% of expected price"
                    )
```

Odoo

Odoo

5. View Development

Creating XML views

Tree/List View:

xml

```
<record id="property_view_tree" model="ir.ui.view">
    <field name="name">estate.property.list</field>
    <field name="model">estate.property</field>
    <field name="arch" type="xml">
        <tree string="Properties">
            <field name="name"/>
            <field name="postcode"/>
            <field name="expected_price"/>
            <field name="state"/>
        </tree>
    </field>
</record>
```

Form View:

```
xml

<record id="property_view_form" model="ir.ui.view">
  <field name="name">estate.property.form</field>
  <field name="model">estate.property</field>
  <field name="arch" type="xml">
    <form string="Property">
      <header>
        <button name="action_sold" type="object" string="Mark as Sold"
          states="new,offer_received,offer_accepted" class="btn-primary"/>
        <field name="state" widget="statusbar"/>
      </header>
      <sheet>
        <div class="oe_title">
          <h1><field name="name"/></h1>
        </div>
        <group>
          <group>
            <field name="property_type_id"/>
            <field name="postcode"/>
            <field name="date_availability"/>
          </group>
          <group>
            <field name="expected_price"/>
            <field name="selling_price"/>
          </group>
        </group>
        <notebook>
          <page string="Description">
            <field name="description"/>
          </page>
          <page string="Offers">
            <field name="offer_ids"/>
          </page>
        </notebook>
      </sheet>
    </form>
  </field>
</record>
```

javascript

```
/** @odoo-module */

import { Component, useState } from "@odoo/owl";
import { registry } from "@web/core/registry";

export class PropertyDashboard extends Component {
    static template = xml`
        <div class="property-dashboard">
            <h3>Property Statistics</h3>
            <p>Total Properties: <t t-esc="state.count"/></p>
            <button t-on-click="refresh" class="btn btn-primary">Refresh</button>
        </div>
    `;

    setup() {
        this.state = useState({ count: 0 });
        this.loadData();
    }

    async loadData() {
        const result = await this.env.services.rpc({
            model: 'estate.property',
            method: 'search_count',
            args: [[]],
        });
        this.state.count = result;
    }

    refresh() {
        this.loadData();
    }
}

registry.category("actions").add("property_dashboard", PropertyDashboard);
```

QWeb templates for reports

xml

```
<template id="report_property_template">
  <t t-call="web.html_container">
    <t t-foreach="docs" t-as="o">
      <t t-call="web.external_layout">
        <div class="page">
          <h2>Property Report</h2>
          <div class="row">
            <div class="col-6">
              <strong>Property:</strong> <span t-field="o.name"/>
            </div>
            <div class="col-6">
              <strong>Expected Price:</strong>
              <span t-field="o.expected_price"
                    t-options="{ 'widget': 'monetary', 'display_currency': o.curre
            </div>
          </div>

          <h3>Offers</h3>
          <table class="table table-sm">
            <thead>
              <tr>
                <th>Partner</th>
                <th>Price</th>
                <th>Status</th>
              </tr>
            </thead>
            <tbody>
              <t t-foreach="o.offer_ids" t-as="offer">
                <tr>
                  <td><span t-field="offer.partner_id.name"/></td>
                  <td><span t-field="offer.price"/></td>
                  <td><span t-field="offer.status"/></td>
                </tr>
              </t>
            </tbody>
          </table>
        </div>
      </t>
    </t>
  </template>
```

6. Controller Development

HTTP controllers and routing

python

```
from odoo import http
from odoo.http import request
import json
```

```
class PropertyController(http.Controller):
```

```
    @http.route('/properties', type='http', auth='public', website=True)
```

```
    def property_list(self, **kw):
```

```
        """Display property listing page"""
```

```
        properties = request.env['estate.property'].sudo().search([
            ('state', '!=', 'sold')
        ])
```

```
    )
```

```
    return request.render('estate.property_listing_template', {
        'properties': properties
    })
```

```
    @http.route('/property/<int:property_id>', type='http', auth='public', website=True)
```

```
    def property_detail(self, property_id, **kw):
```

```
        """Display property detail page"""
```

```
        property_rec = request.env['estate.property'].sudo().browse(property_id)
```

```
        if not property_rec.exists():
```

```
            return request.not_found()
```

```
        return request.render('estate.property_detail_template', {
            'property': property_rec
        })
```

Odoo +2

REST API implementation


```

class PropertyRestController(http.Controller):

    @http.route('/api/v1/properties', type='http', auth='user',
                methods=['GET'], csrf=False)
    def get_properties_rest(self, **kw):
        """GET /api/v1/properties - List all properties"""
        try:
            limit = int(kw.get('limit', 20))
            offset = int(kw.get('offset', 0))

            properties = request.env['estate.property'].search([],
                                                                limit=limit,
                                                                offset=offset)

            data = {
                'properties': [{
                    'id': prop.id,
                    'name': prop.name,
                    'expected_price': prop.expected_price,
                    'state': prop.state,
                } for prop in properties],
                'pagination': {
                    'limit': limit,
                    'offset': offset,
                    'total': request.env['estate.property'].search_count([])
                }
            }

            return request.make_response(
                json.dumps(data),
                headers={'Content-Type': 'application/json'})
        except Exception as e:
            return request.make_response(
                json.dumps({'error': str(e)}),
                status=500,
                headers={'Content-Type': 'application/json'})

    @http.route('/api/v1/properties', type='http', auth='user',
                methods=['POST'], csrf=False)
    def create_property_rest(self, **kw):
        """POST /api/v1/properties - Create new property"""
        try:
            data = json.loads(request.httprequest.data.decode('utf-8'))

```

```

property_rec = request.env['estate.property'].create({
    'name': data.get('name'),
    'expected_price': data.get('expected_price'),
    'bedrooms': data.get('bedrooms'),
    'property_type_id': data.get('property_type_id'),
})

return request.make_response(
    json.dumps({'id': property_rec.id, 'name': property_rec.name}),
    status=201,
    headers={'Content-Type': 'application/json'})
except Exception as e:
    return request.make_response(
        json.dumps({'error': str(e)}),
        status=400,
        headers={'Content-Type': 'application/json'})

```

[GitHub](#)
[Odoo](#)

WebSocket support for real-time features

python

```

class Property(models.Model):
    _name = 'estate.property'
    _inherit = ['estate.property', 'bus.bus']

@api.model
def create(self, vals):
    """Send real-time notification when property is created"""
    property_rec = super().create(vals)

    # Send WebSocket notification
    self.env['bus.bus']._sendmany([
        (self.env.user.partner_id, 'property_created', {
            'id': property_rec.id,
            'name': property_rec.name,
            'message': f'New property "{property_rec.name}" has been created'
        })
    ])

    return property_rec

```

[Odoo +2](#)

7. Business Logic Implementation

Workflow and state management

python

```
class PurchaseWorkflow(models.Model):
    _name = 'purchase.order'

    state = fields.Selection([
        ('draft', 'Draft'),
        ('sent', 'RFQ Sent'),
        ('purchase', 'Purchase Order'),
        ('done', 'Done'),
        ('cancel', 'Cancelled')
    ], default='draft')

    approval_state = fields.Selection([
        ('pending', 'Pending Approval'),
        ('manager_approved', 'Manager Approved'),
        ('finance_approved', 'Finance Approved'),
        ('rejected', 'Rejected'),
    ])

    def action_confirm(self):
        if self.amount_total < 1000:
            self.state = 'purchase'
        else:
            self.approval_state = 'pending'
            self._send_approval_request()

    def _send_approval_request(self):
        if self.amount_total > 10000:
            # Requires both manager and finance approval
            self._notify_managers()
            self._notify_finance_team()
        else:
            # Only manager approval needed
            self._notify_managers()
```

Automated actions and scheduled jobs

Cron Job Example:

xml

```
<record id="cron_process_expired_properties" model="ir.cron">
  <field name="name">Process Expired Properties</field>
  <field name="model_id" ref="model_estate_property"/>
  <field name="state">code</field>
  <field name="code">model._process_expired_properties()</field>
  <field name="interval_number">1</field>
  <field name="interval_type">days</field>
  <field name="numercall">-1</field>
  <field name="active" eval="True"/>
</record>
```

Cybrosys +3

Python Implementation:

python

```
def _process_expired_properties(self):
    expired_properties = self.search([
        ('date_availability', '<', fields.Date.today()),
        ('state', '=', 'new')
    ])

    for prop in expired_properties:
        prop.message_post(body="Property listing has expired")
        prop.state = 'expired'
```

Email templates and notifications

xml

```
<record id="email_template_property_offer" model="mail.template">
  <field name="name">Property Offer Notification</field>
  <field name="model_id" ref="model_estate_property_offer"/>
  <field name="subject">New offer on ${object.property_id.name}</field>
  <field name="body_html" type="html">
    <div style="margin: 0px; padding: 0px;">
      <p>Hello ${object.property_id.user_id.name},</p>
      <p>A new offer has been received for your property
        <strong>${object.property_id.name}</strong>:</p>
      <ul>
        <li>Offer Amount: ${object.price}</li>
        <li>From: ${object.partner_id.name}</li>
        <li>Validity: ${object.validity} days</li>
      </ul>
      <p>Best regards,<br/>The Real Estate Team</p>
    </div>
  </field>
</record>
```

8. Security Implementation

Access rights configuration

csv

```
id,name,model_id:id,group_id:id,perm_read,perm_write,perm_create,perm_unlink
access_property_user,property.user,model_estate_property,base.group_user,1,1,1,0
access_property_manager,property.manager,model_estate_property,estate.group_property_manager,1,
access_property_public,property.public,model_estate_property,,1,0,0,0
```



Record rules implementation

xml

```
<!-- Users can only see their own properties -->
<record id="rule_property_user" model="ir.rule">
    <field name="name">User Property Rule</field>
    <field name="model_id" ref="model_estate_property"/>
    <field name="groups" eval="[(4, ref('base.group_user')))]"/>
    <field name="domain_force">[( 'user_id', '=', user.id)]</field>
</record>

<!-- Managers can see all properties in their company -->
<record id="rule_property_manager" model="ir.rule">
    <field name="name">Manager Property Rule</field>
    <field name="model_id" ref="model_estate_property"/>
    <field name="groups" eval="[(4, ref('estate.group_property_manager')))]"/>
    <field name="domain_force">[ '|', ('company_id', '=', False),
                                ('company_id', 'in', company_ids)]</field>
</record>
```

Odoo

Odoo

Field-level security

python

```
class EstateProperty(models.Model):
    _name = 'estate.property'

    # Only managers can see financial data
    cost_price = fields.Float('Cost Price', groups='estate.group_property_manager')
    margin = fields.Float('Profit Margin', groups='estate.group_property_manager')

    # Sensitive information for specific groups
    internal_notes = fields.Text('Internal Notes',
                                groups='estate.group_property_manager')
```

9. Data Management

CSV and XML data imports

CSV Import Example:

csv

```
id,name,postcode,expected_price,bedrooms,garden,garden_area
property_1,"Beautiful House","12345",250000,3,True,100
property_2,"City Apartment","67890",180000,2,False,0
```

XML Data File:

xml

```
<?xml version="1.0" encoding="UTF-8"?>
<odoo>
  <data noupdate="1">
    <record id="property_type_house" model="estate.property.type">
      <field name="name">House</field>
      <field name="sequence">1</field>
    </record>

    <record id="property_type_apartment" model="estate.property.type">
      <field name="name">Apartment</field>
      <field name="sequence">2</field>
    </record>
  </data>
</odoo>
```

Odoo Odoo

Data migration between versions

python

```
# migrations/18.0.1.1.0/pre-migrate.py
def migrate(cr, version):
    """Pre-migration script"""
    # Rename column before module update
    cr.execute("""
        ALTER TABLE estate_property
        RENAME COLUMN old_field TO new_field
    """)

# migrations/18.0.1.1.0/post-migrate.py
def migrate(cr, version):
    """Post-migration script"""
    # Update data after module update
    cr.execute("""
        UPDATE estate_property
        SET state = 'new'
        WHERE state IS NULL
    """)
```

Odoo Odoo

Database backup strategies

python

```
#!/bin/bash
```

```
# backup_odoo.sh
```

```
BACKUP_DIR="/backups/odoo"
```

```
DATE=$(date +%Y%m%d_%H%M%S)
```

```
DB_NAME="production_db"
```

```
# Create backup directory
```

```
mkdir -p $BACKUP_DIR
```

```
# Backup database
```

```
pg_dump -h localhost -U odoo -d $DB_NAME > $BACKUP_DIR/db_$DATE.sql
```

```
# Backup filestore
```

```
tar -czf $BACKUP_DIR/filestore_$DATE.tar.gz \  
    /var/lib/odoo/.local/share/Odoo/filestore/$DB_NAME
```

```
# Keep only last 7 days of backups
```

```
find $BACKUP_DIR -type f -mtime +7 -delete
```

10. Testing and Debugging Best Practices

Unit testing framework


```

from odoo.tests.common import TransactionCase
from odoo.exceptions import UserError
from odoo.tests import tagged

@tagged('post_install', '-at_install')
class TestEstateProperty(TransactionCase):

    @classmethod
    def setUpClass(cls):
        super().setUpClass()

        # Create test data
        cls.property_type = cls.env['estate.property.type'].create({
            'name': 'Test House Type'
        })

        cls.test_property = cls.env['estate.property'].create({
            'name': 'Test Property',
            'expected_price': 100000,
            'property_type_id': cls.property_type.id,
        })

    def test_01_property_price_validation(self):
        """Test that selling price validation works correctly"""
        with self.assertRaises(UserError):
            # Should fail: selling price < 90% of expected price
            self.test_property.write({'selling_price': 80000})

    def test_02_property_state_workflow(self):
        """Test property state transitions"""
        # Test initial state
        self.assertEqual(self.test_property.state, 'new')

        # Create offer
        offer = self.env['estate.property.offer'].create({
            'property_id': self.test_property.id,
            'partner_id': self.env.user.partner_id.id,
            'price': 95000,
        })

        # Accept offer
        offer.action_accept()
        self.assertEqual(self.test_property.state, 'offer_accepted')

        # Mark as sold

```

```
self.test_property.action_sold()  
self.assertEqual(self.test_property.state, 'sold')
```

Odoo +3

Integration testing

python

```
from odoo.tests.common import HttpCase
```

```
@tagged('post_install', '-at_install')
```

```
class TestPropertyWebsite(HttpCase):
```

```
    def test_01_property_listing_page(self):
```

```
        """Test property listing page renders correctly"""
```

```
        response = self.url_open('/properties')
```

```
        self.assertEqual(response.status_code, 200)
```

```
        self.assertIn(b'Properties', response.content)
```

```
    def test_02_property_tour(self):
```

```
        """Test property creation tour"""
```

```
        self.start_tour("/web", 'property_creation_tour', login='admin')
```

Odoo

Odoo

Debugging techniques

python

Enable debugging in code

import logging

_logger = logging.getLogger(__name__)

class EstateProperty(models.Model):

_name = 'estate. property'

def action_confirm(self):

_logger.info("Confirming property %s", self.name)

Add breakpoint for debugging

import pdb; pdb.set_trace()

Or use ipdb for better debugging

import ipdb; ipdb.set_trace()

try:

result = super().action_confirm()

_logger.info("Property %s confirmed successfully", self.name)

return result

except Exception as e:

_logger.error("Error confirming property %s: %s", self.name, e)

raise

Odoo Odoo

Running Tests:

bash

Run all tests for a module

odoo-bin --test-enable -d test_db -i estate_management

Run specific test tags

odoo-bin --test-tags /estate_management:TestEstateProperty -d test_db

Enable debug mode

odoo-bin --dev=all -d database_name

11. Deployment and Production Considerations

Production server configuration

Nginx Configuration:


```
upstream odoo {
    server 127.0.0.1:8069;
}

upstream odoochat {
    server 127.0.0.1:8072;
}

server {
    listen 443 ssl http2;
    server_name odoo.example.com;

    # SSL Configuration
    ssl_certificate /path/to/certificate.crt;
    ssl_certificate_key /path/to/private.key;
    ssl_protocols TLSv1.2 TLSv1.3;

    # Performance settings
    proxy_read_timeout 720s;
    proxy_connect_timeout 720s;
    proxy_send_timeout 720s;

    # Headers
    proxy_set_header X-Forwarded-Host $host;
    proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
    proxy_set_header X-Forwarded-Proto $scheme;

    location / {
        proxy_pass http://odoo;
    }

    location /longpolling {
        proxy_pass http://odoochat;
    }

    # Static file caching
    location ~* /web/static/ {
        proxy_cache_valid 200 60m;
        proxy_buffering on;
        expires 864000;
        proxy_pass http://odoo;
    }

    # Gzip compression
    gzip on;
    gzip_types text/css text/plain text/xml application/xml
```

```
        application/json application/javascript;
    client_max_body_size 500M;
}
```

PostgreSQL optimization

```
ini

# postgresql.conf for production
shared_buffers = 3072MB          # 20% of total RAM
effective_cache_size = 8192MB    # 50% of total RAM
work_mem = 256MB
maintenance_work_mem = 512MB
wal_buffers = 16MB
checkpoint_segments = 32
random_page_cost = 1.1           # For SSD storage

# Enable slow query logging
log_min_duration_statement = 2000 # Log queries > 2 seconds
```

Multi-worker setup

ini

```
# odoo.conf for production
[options]
# Workers configuration
workers = 8 # (CPU cores * 2) + 1
max_cron_threads = 2

# Memory limits
limit_memory_hard = 1677721600 # 1.6GB per worker
limit_memory_soft = 629145600 # 640MB per worker

# Request limits
limit_request = 8192
limit_time_cpu = 600
limit_time_real = 1200

# Database settings
db_host = localhost
db_port = 5432
db_user = odoo
db_password = secure_password

# Logging
logfile = /var/log/odoo/odoo.log
log_level = info
```

12. Integration with External Systems

REST API integration patterns

python

```
import requests
```

```
import json
```

```
class ExternalAPIIntegration:
```

```
    def __init__(self, base_url, api_key):
```

```
        self.base_url = base_url
```

```
        self.headers = {
```

```
            'Authorization': f'Bearer {api_key}',
```

```
            'Content-Type': 'application/json'
```

```
        }
```

```
    def sync_customer_data(self, partner):
```

```
        """Sync customer data with external CRM"""
```

```
        endpoint = f"{self.base_url}/customers"
```

```
        data = {
```

```
            'name': partner.name,
```

```
            'email': partner.email,
```

```
            'phone': partner.phone,
```

```
            'odoo_id': partner.id
```

```
        }
```

```
        response = requests.post(endpoint,
```

```
                                json=data,
```

```
                                headers=self.headers)
```

```
        if response.status_code == 201:
```

```
            external_id = response.json().get('id')
```

```
            partner.external_crm_id = external_id
```

```
        else:
```

```
            raise Exception(f"API Error: {response.text}")
```

XML-RPC usage

python

```
import xmlrpc.client

# Connect to Odoo
url = 'http://localhost:8069'
db = 'production_db'
username = 'admin'
password = 'admin'

# Authenticate
common = xmlrpc.client.ServerProxy(f'{url}/xmlrpc/2/common')
uid = common.authenticate(db, username, password, {})

# Execute operations
models = xmlrpc.client.ServerProxy(f'{url}/xmlrpc/2/object')

# Search partners
partner_ids = models.execute_kw(
    db, uid, password,
    'res.partner', 'search',
    [[['is_company', '=', True]])
)

# Read partner data
partners = models.execute_kw(
    db, uid, password,
    'res.partner', 'read',
    [partner_ids],
    {'fields': ['name', 'email', 'phone']}
)
```

Third-party service integrations

Payment Gateway Integration:

python

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

class PaymentAcquirer(models.Model):
    _inherit = 'payment.acquirer'

    provider = fields.Selection(selection_add=[('stripe', 'Stripe')])
    stripe_secret_key = fields.Char('Secret Key', groups='base.group_system')
    stripe_publishable_key = fields.Char('Publishable Key')

    @api.model
    def _stripe_create_payment_intent(self, amount, currency):
        stripe.api_key = self.stripe_secret_key

        return stripe.PaymentIntent.create(
            amount=int(amount * 100), # Convert to cents
            currency=currency,
            payment_method_types=['card'],
        )
```

13. Performance Optimization

Database query optimization

python

Efficient ORM usage

```
class EstateProperty(models.Model):
    _name = 'estate.property'

    def get_property_statistics(self):
        # Bad: Multiple queries
        # for prop in self:
        #     offers = self.env['estate.property.offer'].search([
        #         ('property_id', '=', prop.id)
        #     ])

        # Good: Single query with grouping
        self.env.cr.execute("""
            SELECT property_id, COUNT(*), AVG(price)
            FROM estate_property_offer
            WHERE property_id IN %s
            GROUP BY property_id
        """, (tuple(self.ids),))

        stats = dict(
            (row[0], {'count': row[1], 'avg_price': row[2]})
            for row in self.env.cr.fetchall()
        )

        return stats
```

Caching strategies

python

```
from odoo import tools

class ProductPricing(models.Model):
    _name = 'product.pricing'

    @tools.ormcache('self.id', 'pricelist_id', 'quantity')
    def get_product_price(self, pricelist_id, quantity=1.0):
        """Cached price computation"""
        pricelist = self.env['product.pricelist'].browse(pricelist_id)
        return pricelist._compute_price_rule(
            [(self, quantity, False)]
        )[self.id][0]

    def clear_price_cache(self):
        """Clear cache when prices change"""
        self.get_product_price.clear_cache(self)
```

Asset optimization

python

```
# Manifest asset configuration
{
    'name': 'Optimized Module',
    'assets': {
        'web.assets_backend': [
            ('include', 'estate_management/static/src/css/estate.css'),
            ('include', 'estate_management/static/src/js/estate.js'),
        ],
        'web.assets_backend_prod_only': [
            ('replace', 'estate_management/static/src/js/estate.js',
             'estate_management/static/src/js/estate.min.js'),
        ],
    },
}
```

14. Version Control and Collaboration Workflows

Git workflow for Odoo

bash

Feature branch workflow

git checkout 18.0

git pull origin 18.0

git checkout -b 18.0-feature-property-search

Make changes and commit

git add .

git commit -m "[IMP] estate: improve property search functionality

- Add advanced filters for property search
- Implement full-text search on property descriptions
- Add search by price range

Task: TASK-123"

Push and create pull request

git push origin 18.0-feature-property-search

CI/CD pipeline configuration


```
# .github/workflows/odoo-ci.yml
name: Odoo CI/CD Pipeline
on:
  push:
    branches: [main, 18.0]
  pull_request:
    branches: [main]

jobs:
  test:
    runs-on: ubuntu-latest
    services:
      postgres:
        image: postgres:13
        env:
          POSTGRES_PASSWORD: odoo
        options: >-
          --health-cmd pg_isready
          --health-interval 10s

    steps:
      - uses: actions/checkout@v2

      - name: Set up Python
        uses: actions/setup-python@v2
        with:
          python-version: 3.11

      - name: Install dependencies
        run: |
          pip install -r requirements.txt
          pip install flake8 coverage

      - name: Lint with flake8
        run: |
          flake8 . --config=.flake8

      - name: Run tests
        run: |
          ./odoo-bin -d test_db -i estate_management \
            --test-enable --stop-after-init

      - name: Generate coverage report
        run: |
          coverage run ./odoo-bin -d test_db -i estate_management \
```



```
--test-enable --stop-after-init
coverage report
```

Team collaboration best practices

python

```
# Code review checklist
```

```
"""
```

Before submitting a pull request:

```
[ ] Code follows Odoo coding standards
[ ] All tests pass
[ ] New features have tests
[ ] Documentation is updated
[ ] Security rules are implemented
[ ] Performance impact considered
[ ] Migration scripts included if needed
```

```
"""
```

```
# Example .flake8 configuration
```

```
[flake8]
```

```
max-line-length = 120
```

```
exclude = .git,__pycache__,migrations
```

```
ignore = E203,W503
```

15. Common Development Patterns and Best Practices

Design patterns implementation

Singleton Pattern:

python

```
class SystemConfiguration(models.Model):
    _name = 'system.configuration'
    _description = 'System Configuration'

    @api.model
    def get_instance(self):
        config = self.search([], limit=1)
        if not config:
            config = self.create({
                'name': 'Default Configuration'
            })
        return config
```

Factory Pattern:

python

```
class ReportFactory(models.Model):
    _name = 'report.factory'

    @api.model
    def create_report(self, report_type, data):
        report_map = {
            'sales': 'sale.report',
            'inventory': 'stock.report',
            'financial': 'account.report'
        }

        model_name = report_map.get(report_type)
        if not model_name:
            raise ValueError(f"Unknown report type: {report_type}")

        return self.env[model_name].create_report(data)
```

Error handling patterns

python

```
from odoo.exceptions import UserError, ValidationError
import logging

_logger = logging.getLogger(__name__)

class RobustPropertyModel(models.Model):
    _name = 'robust.property'

    def safe_operation(self):
        """Example of robust error handling"""
        try:
            # Validate input
            self._validate_operation_data()

            # Perform operation
            result = self._perform_critical_operation()

            # Log success
            _logger.info("Operation completed successfully for %s", self.name)

            return result

        except ValidationError as e:
            # Handle validation errors
            _logger.warning("Validation error: %s", e)
            self.message_post(body=f"Validation Error: {e}")
            raise

        except Exception as e:
            # Handle unexpected errors
            _logger.error("Unexpected error in operation: %s", e, exc_info=True)
            self.message_post(
                body="An unexpected error occurred. Please contact support.",
                message_type='notification'
            )
            raise UserError("Operation failed. Please try again later.")

        finally:
            # Cleanup operations
            self._cleanup_temporary_data()
```

Performance best practices

python

```
class OptimizedModel(models.Model):
    _name = 'optimized.model'

    def batch_process_records(self, record_ids):
        """Process large datasets efficiently"""
        batch_size = 100

        for i in range(0, len(record_ids), batch_size):
            batch_ids = record_ids[i:i + batch_size]
            records = self.browse(batch_ids)

            # Process batch
            self._process_batch(records)

            # Commit to avoid long transactions
            self.env.cr.commit()

            # Clear cache to free memory
            self.env.invalidate_all()

    def optimized_search(self, domain):
        """Use read_group for aggregated data"""
        # Instead of searching and computing
        # records = self.search(domain)
        # total = sum(records.mapped('amount'))

        # Use read_group
        result = self.read_group(
            domain,
            ['amount:sum'],
            []
        )
        return result[0]['amount'] if result else 0
```

Common pitfalls to avoid

python

```
# PITFALL 1: Modifying Loop variable
# Bad
for record in self:
    record = record.with_context(lang='en_US') # Don't do this!

# Good
for record in self:
    record_en = record.with_context(lang='en_US')

# PITFALL 2: Not using batch operations
# Bad
for line in order.order_line:
    line.write({'discount': 10}) # Multiple writes

# Good
order.order_line.write({'discount': 10}) # Single write

# PITFALL 3: Forgetting to handle multi-company
# Bad
domain = [('state', '=', 'draft')]

# Good
domain = [
    ('state', '=', 'draft'),
    ('company_id', 'in', self.env.companies.ids)
]

# PITFALL 4: Inefficient related field access
# Bad
for order in orders:
    print(order.partner_id.country_id.name) # Multiple queries

# Good
orders.mapped('partner_id.country_id.name') # Prefetched
```

Conclusion

This comprehensive guide covers all aspects of Odoo 18.0 development, from initial setup through deployment and optimization. By following these guidelines and best practices, developers can build robust, scalable, and maintainable Odoo applications.

Remember to:

- Always follow Odoo coding standards

- Write comprehensive tests for your code
- Document your modules thoroughly
- Consider security at every level
- Optimize for performance from the start
- Use version control effectively
- Collaborate with your team using established workflows

For the latest updates and additional resources, refer to the official Odoo 18.0 documentation at <https://www.odoo.com/documentation/18.0/developer.html>.