**OdooSkills.com**

Avançons ensemble !

# GUIDE TECHNIQUE FRAMEWORK ODOO 19

## De l'architecture au premier module fonctionnel

| 6 | 19 | 100% |
|---|---|---|
| Chapitres | Version Odoo | Code testé |

Par BENHAMIDA Mustapha

Consultant Technico-Fonctionnel Odoo — Fondateur ADICOPS

**Édition 2026 — Odoo 19 Community**

# Sommaire



Ce guide te donnera les fondations techniques pour comprendre, créer et déployer un module Odoo 19. À la fin de ta lecture, tu auras un module fonctionnel et les connaissances pour aller plus loin.

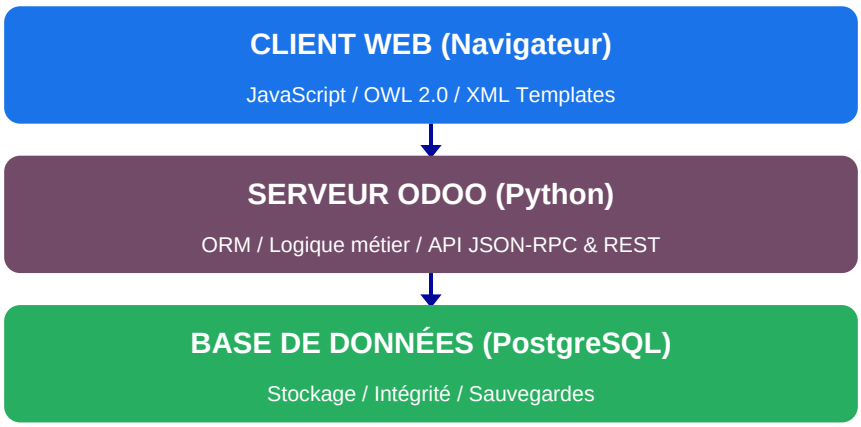
# 01 Architecture technique d'Odoo 19

## 1.1 — Le pattern MVC revisité par Odoo

Odoo s'appuie sur une variante du pattern MVC (Modèle-Vue-Contrôleur). Contrairement au MVC classique, les vues sont **déclaratives** (XML), l'ORM gère automatiquement la persistance (pas de SQL manuel), et le frontend utilise le framework réactif **OWL 2.0** (Odoo Web Library).

| Couche | Rôle dans Odoo | Technologie |
|---|---|---|
| Models (M) | Logique métier + accès données | Python + ORM Odoo |
| Views (V) | Interface utilisateur | XML (définition) + QWeb + OWL 2.0 |
| Controllers (C) | Routes HTTP, API | Python (classes Controller) |

## 1.2 — Architecture trois-tiers

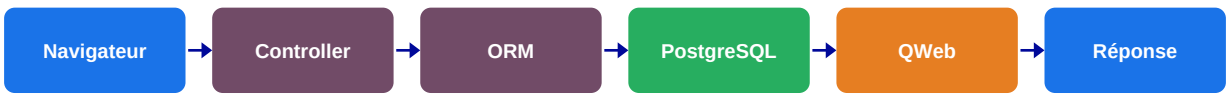



**Couche Client** — L'interface s'exécute dans le navigateur. Depuis Odoo 18, le framework JS a été remplacé par OWL 2.0, un framework réactif moderne.

**Couche Serveur** — Le cœur d'Odoo : logique métier, moteur ORM. Les API JSON-RPC et REST permettent l'intégration externe.

**Couche Base de données** — PostgreSQL est le seul SGBD supporté.

## 1.3 — Cycle de vie d'une requête




Chaque interaction suit ce cycle : navigateur → Controller → ORM → PostgreSQL → QWeb → réponse.

## 1.4 — Structure d'une instance Odoo

```
odoo/ # Code source Odoo (core)
■■■ odoo/ # Framework Python
■ ■■■ addons/ # Modules de base (base, mail, web...)
■ ■■■ tools/ # Utilitaires framework
■■■ addons/ # Modules officiels (sale, stock, account...)
■■■ setup/ # Scripts d'installation

custom-addons/ # Tes modules personnalisés
■■■ mon_module/
■■■ autre_module/

config/
■■■ odoo.conf # Configuration serveur
```

■ **Astuce** — Odoo 19 contient plus de 600 modules officiels. Le core (**base**) fait environ 50 000 lignes de Python. Quand tu crées un module custom, tu hérites de toute cette puissance.

02

# Anatomie d'un module Odoo 19

## 2.1 — Arborescence type

```
mon_module/
■■■ __init__.py # Import des sous-packages
■■■ __manifest__.py # Métadonnées du module
■■■ models/
■ ■■■ __init__.py
■ ■■■ mon_modele.py # Définition des modèles
■■■ views/
■ ■■■ mon_modele_views.xml # Vues (form, list, menus)
■■■ security/
■ ■■■ ir.model.access.csv # Droits d'accès
■■■ data/
■ ■■■ data.xml # Données initiales
■■■ static/
■ ■■■ description/
■ ■■■ icon.png # Icône du module (256×256)
■■■ README.md
```

## 2.2 — Le __manifest__.py expliqué

```
{
    'name': 'Mon Module',
    'version': '19.0.1.0.0',
    'category': 'Services',
    'summary': 'Description courte',
    'author': 'Mon Entreprise',
    'website': 'https://monsite.com',
    'license': 'LGPL-3',
    'depends': ['base', 'mail'],
    'data': [
        'security/ir.model.access.csv',  # Sécurité EN PREMIER
        'views/mon_modele_views.xml',
        'data/data.xml',
    ],
    'installable': True,
    'application': True,
    'auto_install': False,
}
```

■■ **Piège** — L'ordre dans **'data'** est critique : charge toujours le fichier de sécurité en premier.

## 2.3 — Module complet « Bibliothèque » en 4 fichiers

Voici un module complet et fonctionnel pour Odoo 19. Tu peux le copier tel quel.

**Fichier 1 : __manifest__.py**

```
{
    'name': 'Bibliothèque',
    'version': '19.0.1.0.0',
    'category': 'Services',
```

```
    'summary': 'Gestion simple de livres',
    'author': 'OdooSkills',
    'license': 'LGPL-3',
    'depends': ['base'],
    'data': [
        'security/ir.model.access.csv',
        'views/book_views.xml',
    ],
    'installable': True,
    'application': True,
}
```

**Fichier 2 : models/book.py**

```
from odoo import models, fields

class Book(models.Model):
    _name = 'library.book'
    _description = 'Livre'

    name = fields.Char(required=True)
    author_name = fields.Char()
    isbn = fields.Char()
    publication_date = fields.Date()
    pages = fields.Integer()
    category = fields.Selection([
        ('fiction', 'Fiction'),
        ('tech', 'Technique'),
        ('business', 'Business'),
    ])
    active = fields.Boolean(default=True)
```

■ **Astuce** — En Odoo 19, le paramètre **string=** est optionnel. Odoo génère le libellé à partir du nom du champ. Pour un libellé en français, utilise la traduction via les fichiers **i18n/**.

**Fichier 3 : security/ir.model.access.csv**

```
id,name,model_id:id,group_id:id,perm_read,perm_write,perm_create,perm_unlink
access_library_book_user,library.book.user,model_library_book,base.group_user,1,1,1,1
```

### Fichier 4 : views/book_views.xml

```xml
<?xml version="1.0" encoding="utf-8"?>
<odoo>
    <!-- Vue formulaire -->
    <record id="library_book_view_form" model="ir.ui.view">
        <field name="name">library.book.form</field>
        <field name="model">library.book</field>
        <field name="arch" type="xml">
            <form>
                <sheet>
                    <group>
                        <group>
                            <field name="name"/>
                            <field name="author_name"/>
                            <field name="isbn"/>
                        </group>
                        <group>
                            <field name="publication_date"/>
                            <field name="pages"/>
                            <field name="category"/>
                        </group>
                    </group>
                </sheet>
            </form>
        </field>
    </record>

    <!-- Vue liste -->
    <record id="library_book_view_list" model="ir.ui.view">
        <field name="name">library.book.list</field>
        <field name="model">library.book</field>
        <field name="arch" type="xml">
            <list>
                <field name="name"/>
                <field name="author_name"/>
                <field name="category"/>
                <field name="publication_date"/>
            </list>
        </field>
    </record>

    <!-- Action -->
    <record id="library_book_action" model="ir.actions.act_window">
        <field name="name">Livres</field>
        <field name="res_model">library.book</field>
        <field name="view_mode">list,form</field>
    </record>

    <!-- Menu -->
    <menuitem id="library_menu_root" name="Bibliothèque" sequence="10"/>
    <menuitem id="library_menu_books" name="Livres"
              parent="library_menu_root"
              action="library_book_action" sequence="10"/>
</odoo>
```

### N'oublie pas les __init__.py !

```python
# __init__.py (racine du module)
from . import models

# models/__init__.py
from . import book
```

■ **Astuce** — Ce module est 100% fonctionnel sur Odoo 19 CE. Place le dossier dans ton répertoire custom addons, mets à jour la liste des modules et installe « Bibliothèque ».

# 03 L'ORM Odoo en 10 minutes

## 3.1 — Les 3 types de modèles

| Type | Usage | Persistance |
|---|---|---|
| models.Model | Données métier (clients, factures...) | Table PostgreSQL permanente |
| models.TransientModel | Assistants (wizards) | Table vidée automatiquement |
| models.AbstractModel | Classe de base partagée (mixin) | Pas de table propre |

## 3.2 — Champs essentiels

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

class Exemple(models.Model):
    _name = 'mon.exemple'
    _description = 'Exemple de modèle'

    # Champs simples
    name = fields.Char(required=True)
    description = fields.Text()
    montant = fields.Float(digits=(10, 2))
    quantite = fields.Integer(default=1)
    actif = fields.Boolean(default=True)
    date_debut = fields.Date()
    date_heure = fields.Datetime()
    contenu = fields.Html()

    # Sélection
    etat = fields.Selection([
        ('brouillon', 'Brouillon'),
        ('confirme', 'Confirmé'),
        ('termine', 'Terminé'),
    ], default='brouillon')

    # Relations
    partenaire_id = fields.Many2one('res.partner')
    ligne_ids = fields.One2many('mon.exemple.ligne', 'exemple_id')
    tag_ids = fields.Many2many('mon.exemple.tag')

    # Champ calculé
    nom_complet = fields.Char(compute='_compute_nom_complet')

    @api.depends('name', 'partenaire_id.name')
    def _compute_nom_complet(self):
        for record in self:
            partner = record.partenaire_id.name or 'Sans client'
            record.nom_complet = f'{record.name} — {partner}'
```

## 3.3 — Les 3 types de relations

| | | |
|---|---|---|
| **Many2one** | Commande ■■■■■■■ Client | *(N → 1)* |
| **One2many** | Client ■■■■■■■ Commandes | *(1 → N)* |
| **Many2many** | Produit ■■■■■■■ Tags | *(N ↔ N)* |

Le **Many2one** crée une clé étrangère en base. Le **One2many** est son miroir (champ virtuel). Le **Many2many** crée une table de liaison automatiquement.

## 3.4 — CRUD : les 4 opérations de base

```python
# CRÉER un enregistrement
livre = self.env['library.book'].create({
    'name': 'Odoo 19 Development',
    'author_name': 'OdooSkills',
    'pages': 350,
})

# LIRE / RECHERCHER
livres = self.env['library.book'].search([
    ('category', '=', 'tech'),
    ('pages', '>', 200),
])
for livre in livres:
    print(livre.name, livre.pages)

# MODIFIER
livre.write({'pages': 400, 'category': 'tech'})

# SUPPRIMER
livre.unlink()

# BROWSE (accès par ID)
livre = self.env['library.book'].browse(42)
```

■■ **Piège** — Ne confonds jamais **search()** et **browse()**. search() prend un **domaine** (filtre), browse() prend un **ID**.

# 04 Les vues essentielles

Les vues Form et List ont été présentées au chapitre 2. Voyons les deux autres vues principales.

## 4.1 — Vue Search

```xml
<record id="library_book_view_search" model="ir.ui.view">
    <field name="name">library.book.search</field>
    <field name="model">library.book</field>
    <field name="arch" type="xml">
        <search>
            <field name="name"/>
            <field name="author_name"/>
            <filter name="technique" string="Technique"
                    domain="[('category', '=', 'tech')]"/>
            <filter name="group_category" string="Catégorie"
                    context="{'group_by': 'category'}"/>
        </search>
    </field>
</record>
```

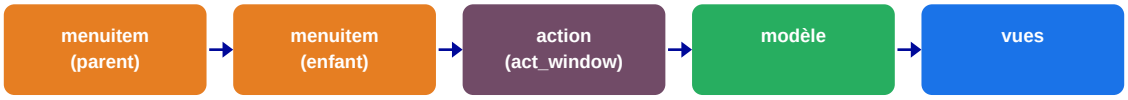
■ **Odoo 19** — En Odoo 19, la balise **<group expand="0">** n'est plus nécessaire dans les vues search. Les filtres group_by vont directement dans **<search>**.

## 4.2 — Vue Kanban

```xml
<record id="library_book_view_kanban" model="ir.ui.view">
    <field name="name">library.book.kanban</field>
    <field name="model">library.book</field>
    <field name="arch" type="xml">
        <kanban>
            <templates>
                <t t-name="card">
                    <field name="name"/>
                    <field name="author_name"/>
                    <field name="category"/>
                </t>
            </templates>
        </kanban>
    </field>
</record>
```

■ **Odoo 19** — Le template kanban s'appelle désormais **t-name="card"** (et non plus **kanban-box**).

## 4.3 — Le câblage : Action → Menu → Vue




L'ordre dans **view_mode** détermine la vue par défaut (la première listée).

## 4.4 — Héritage de vues avec XPath

```xml
<record id="view_partner_form_inherit_library" model="ir.ui.view">
    <field name="name">res.partner.form.inherit.library</field>
    <field name="model">res.partner</field>
    <field name="inherit_id" ref="base.view_partner_form"/>
    <field name="arch" type="xml">
        <xpath expr="//field[@name='phone']" position="after">
            <field name="x_favorite_book"/>
        </xpath>
    </field>
</record>
```

| Position | Effet |
|---|---|
| after | Ajoute le contenu après le nœud ciblé |
| before | Ajoute le contenu avant le nœud ciblé |
| inside | Ajoute à l'intérieur du nœud ciblé |
| replace | Remplace entièrement le nœud ciblé |
| attributes | Modifie les attributs du nœud ciblé |

■ **Odoo 19** — Les expressions conditionnelles remplacent l'ancien **attrs**. Écris **invisible="state == 'draft'"** directement sur le champ.

# 05 Sécurité en 5 minutes

## 5.1 — Groupes de sécurité

En Odoo 19, le système de groupes a évolué avec l'introduction de **res.groups.privilege**. Le champ **category_id** sur res.groups est remplacé par **privilege_id**.

```
<!-- Nouveau modèle v19 : res.groups.privilege -->
<record id="groups_privilege_library" model="res.groups.privilege">
    <field name="name">Bibliothèque</field>
    <field name="sequence">50</field>
</record>

<!-- Groupe utilisateur -->
<record id="group_library_user" model="res.groups">
    <field name="name">Utilisateur Bibliothèque</field>
    <field name="privilege_id" ref="groups_privilege_library"/>
</record>

<!-- Groupe manager (hérite de user) -->
<record id="group_library_manager" model="res.groups">
    <field name="name">Responsable Bibliothèque</field>
    <field name="privilege_id" ref="groups_privilege_library"/>
    <field name="implied_ids" eval="[(4, ref('group_library_user'))]"/>
</record>
```

■ **Odoo 19** — En Odoo 19, **category_id** sur res.groups est remplacé par **privilege_id** pointant vers le nouveau modèle **res.groups.privilege**. Également : **users** → **user_ids** sur res.groups, et **groups_id** → **group_ids** sur res.users.

## 5.2 — ir.model.access.csv ligne par ligne

```
id,name,model_id:id,group_id:id,perm_read,perm_write,perm_create,perm_unlink
access_book_user,Livres Lecture,model_library_book,group_library_user,1,0,0,0
access_book_manager,Livres Complet,model_library_book,group_library_manager,1,1,1,1
```

| Colonne | Signification |
|---|---|
| id | Identifiant XML unique |
| name | Nom descriptif (libre) |
| model_id:id | Modèle ciblé (model_ + _name avec . → _) |
| group_id:id | Groupe concerné (vide = tous les utilisateurs) |
| perm_read/write/create/unlink | 1 = autorisé, 0 = interdit |

## 5.3 — Record Rules

```
<record id="rule_book_own_company" model="ir.rule">
    <field name="name">Livres de ma société</field>
    <field name="model_id" ref="model_library_book"/>
    <field name="domain_force">
```

```
        [('company_id', '=', user.company_id.id)]
    </field>
    <field name="groups" eval="[(4, ref('group_library_user'))]"/>
</record>
```

## 5.4 — Quand utiliser sudo() ?

| Situation | sudo() ? | Pourquoi |
|---|---|---|
| Lire ir.config_parameter | Oui | Paramètres système |
| Générer ir.sequence | Oui | Opération système |
| Créer sale.order | NON | Données métier — respecter les droits |
| Modifier un partenaire | NON | Bypass dangereux des permissions |

■■ **Piège** — **sudo()** contourne toutes les règles de sécurité. Ne l'utilise que pour des opérations système, **jamais** pour des données métier.

06

# Checklist « Mon premier module »

Imprime cette page et coche chaque étape !

- [ ] **1.** Créer le dossier du module dans custom-addons/
- [ ] **2.** Créer __init__.py (racine + sous-dossiers)
- [ ] **3.** Écrire __manifest__.py (name, version, depends, data)
- [ ] **4.** Créer models/ avec ton modèle Python
- [ ] **5.** Créer security/ir.model.access.csv
- [ ] **6.** Créer views/ avec les vues XML (form + list)
- [ ] **7.** Ajouter action + menus dans les vues
- [ ] **8.** Redémarrer Odoo : ./odoo-bin -c config/odoo.conf
- [ ] **9.** Activer le mode développeur (Settings → Activate Developer Mode)
- [ ] **10.** Mettre à jour la liste des modules (Apps → Update Apps List)
- [ ] **11.** Rechercher ton module et l'installer
- [ ] **12.** Vérifier : menu visible, formulaire fonctionnel, données créables

## Commandes terminal de référence

```
# Lancer Odoo
./odoo-bin -c config/odoo.conf

# Installer un module
./odoo-bin -c config/odoo.conf -i mon_module -d ma_base

# Mettre à jour un module (après modification)
./odoo-bin -c config/odoo.conf -u mon_module -d ma_base

# Lancer les tests
./odoo-bin -c config/odoo.conf --test-enable -i mon_module -d test_db --stop-after-init
```

## Les 5 vérifications post-installation

1. Le menu principal apparaît dans la barre latérale
2. La vue liste affiche les colonnes définies
3. Le formulaire s'ouvre et tous les champs sont visibles
4. Tu peux créer, modifier et supprimer un enregistrement
5. Les logs Odoo ne montrent aucun warning XML ou Python

# Tu as les bases — la suite est sur OdooSkills.com

Ce guide t'a donné les fondations pour comprendre et créer un module Odoo 19. Mais le développement Odoo, c'est bien plus que ça : les wizards, les rapports PDF, les vues avancées (Graph, Pivot, Gantt), les composants OWL, les API REST, les tests automatisés...

Tous ces sujets sont couverts en détail dans nos articles gratuits sur le blog.

| Ressource | URL |
|---|---|
| Blog Développement Odoo | odooskills.com/blog/developpement-odoo |
| Blog Fonctionnel Odoo | odooskills.com/blog/fonctionnel-odoo |
| Guide Express Odoo (1er lead magnet) | odooskills.com/guide-express-odoo |
| Newsletter OdooSkills | odooskills.com/guide-technique-odoo |

**Ressources externes recommandées :**

→ Documentation officielle Odoo 19 : odoo.com/documentation/19.0/

→ Code source GitHub : github.com/odoo

→ Plateforme de test Runbot : runbot.odoo.com

**Inscris-toi à la newsletter OdooSkills** pour recevoir un nouvel article chaque semaine, directement dans ta boîte mail.

**odooskills.com/guide-technique-odoo**

# À propos de l'auteur

**BENHAMIDA Mustapha** est consultant technico-fonctionnel Odoo avec plus de 10 ans d'expérience. Fondateur d'**ADICOPS** (consulting SI) et du blog **OdooSkills.com**, il accompagne les entreprises francophones dans leur transformation digitale avec Odoo.

**Site web** : odooskills.com

**Email** : info@odooskills.com

**Société** : ADICOPS — Consulting SI

## « Avançons ensemble ! »