École Nationale des Sciences Appliquées
ENSA Khouribga

GESTION DES GARANTIES ODOO

Module de suivi et d'automatisation

Rapport technique d'implémentation

Encadré par :

Prof. Nidal LMGHARI

Réalisé par :

Yosra EL MIMOUNI

Imane BOUHABBA

Oumaima MELLAH

Année universitaire : 2025-2026

29 décembre 2025

Table des matières

1 Introduction

1.1 Contexte du Projet

Dans un environnement commercial moderne, la gestion efficace des garanties produits constitue un élément crucial de la satisfaction client et de la conformité réglementaire. Ce projet répond au besoin d'automatiser et de centraliser la gestion des garanties au sein d'un système ERP Odoo.

1.2 Objectifs

Les objectifs principaux de ce module sont :

— Centraliser toutes les informations de garantie dans un système unique

— Automatiser le calcul des dates d'expiration

— Faciliter le suivi des réclamations clients

— Améliorer la traçabilité et la conformité

— Réduire les erreurs humaines et le temps de traitement

2 Problématique et Solution

2.1 Le Défi

Le contexte actuel de gestion des garanties présente plusieurs défis majeurs qui impactent l'efficacité opérationnelle :

Problèmes Identifiés

— **Suivi manuel fastidieux** : Les dates d'expiration sont gérées manuellement sur des tableurs dispersés, entraînant une perte de temps considérable

— **Risques d'erreurs** : Le facteur humain introduit un risque élevé d'erreurs et d'oublis critiques pouvant affecter la relation client

— **Manque de visibilité** : Absence de centralisation des réclamations clients, rendant difficile le suivi et l'analyse des tendances

— **Perte d'information** : Dispersion des données entre plusieurs systèmes et fichiers

2.2 La Solution Proposée

Pour répondre à ces défis, nous avons développé un module Odoo intégré offrant une approche moderne et automatisée :

Avantages du Module

- **Centralisation** : Module Odoo intégré pour une gestion centralisée de toutes les garanties
- **Automatisation** : Calcul automatique des échéances en temps réel, éliminant les erreurs manuelles
- **Traçabilité** : Suivi rigoureux du cycle de vie via des statuts clairs et une interface intuitive
- **Satisfaction client** : Amélioration directe de la qualité du service après-vente (SAV)
- **Intégration native** : Liaison directe avec les modules Ventes et Contacts d'Odoo

3 Architecture Technique

Le module repose sur une architecture modulaire suivant les conventions d'Odoo, garantissant maintenabilité et évolutivité.

3.1 Structure des Composants

Composants du Module

1. **MODÈLES (Python)**
   - Définition de l'objet `gestion.garantie`
   - Logique métier et calculs automatisés
   - Validation des données et contraintes
2. **VUES (XML)**
   - Interfaces utilisateur : formulaires détaillés
   - Listes filtrables et recherche avancée
   - Menus d'accès hiérarchisés
3. **DONNÉES (XML)**
   - Configuration des séquences pour numérotation automatique
   - Données initiales et de démonstration
4. **SÉCURITÉ (CSV)**
   - Définition des droits d'accès (ACL)
   - Garantie de l'intégrité des données
   - Gestion des permissions par groupe utilisateur
5. **RESSOURCES STATIQUES**
   - Icônes et éléments visuels
   - Identification rapide du module dans l'interface
6. **MANIFESTE**
   - Métadonnées du module
   - Gestion du versioning
   - Déclaration des fichiers et dépendances

3.2 Dépendances

Le module s'appuie sur les modules Odoo standards :

- `base` : Fonctionnalités de base d'Odoo
- `sale` : Gestion des ventes pour lier les garanties aux commandes
- `product` : Gestion des produits

4 Implémentation Détaillée

4.1 Le Modèle Python

Le cœur du module réside dans le modèle `gestion.garantie` qui hérite de `models.Model`. Voici les éléments clés de l'implémentation :

4.1.1 Champs Principaux

```python
class GestionGarantie(models.Model):
    _name = 'gestion.garantie'
    _description = 'Gestion des Garanties'
    _order = 'create_date desc'

    # Champs de base
    name = fields.Char(
        string='Numéro de Garantie',
        required=True,
        copy=False,
        readonly=True,
        index=True,
        default='/'
    )

    # Relations
    partner_id = fields.Many2one(
        'res.partner',
        string='Client',
        required=True,
        help='Le client concerné par la garantie'
    )
```

FIGURE 1 – Définition des champs du modèle

4.1.2 Génération Automatique de Numéro

La numérotation automatique est un élément essentiel qui garantit l'unicité de chaque garantie :

```python
    Qodo: Test this method
    def _get_or_create_sequence(self):
        """Récupère ou crée la séquence pour les numéros de garantie"""
        IrSequence = self.env['ir.sequence'].sudo()

        # Chercher la séquence existante
        sequence = IrSequence.search([('code', '=', 'gestion.garantie')], limit=1)

        # Si elle n'existe pas, la créer
        if not sequence:
            sequence = IrSequence.create({
                'name': 'Numéro de Garantie',
                'code': 'gestion.garantie',
                'implementation': 'standard',
                'active': True,
                'prefix': 'GAR/%(year)s/',
                'padding': 5,
                'number_next': 1,
                'number_increment': 1,
            })

        return sequence
```

```
Qodo: Test this method
@api.model_create_multi
def create(self, vals_list):
    """Générer automatiquement le numéro de garantie"""
    for vals in vals_list:
        if not vals.get('name') or vals.get('name') == '/':
            # Récupérer ou créer la séquence
            sequence = self._get_or_create_sequence()

            # Générer le numéro
            vals['name'] = sequence._next()

    return super(GestionGarantie, self).create(vals_list)
```

FIGURE 2 – Génération du numéro de garantie

4.1.3 Calcul Automatique de l'Expiration

L'un des points forts du module est le calcul automatique de la date d'expiration :

```
Qodo: Test this method
@api.depends('purchase_date', 'duration_months')
def _compute_expiration_date(self):
    """Calcule la date d'expiration basée sur la date d'achat et la durée."""
    for record in self:
        if record.purchase_date and record.duration_months:
            record.expiration_date = record.purchase_date + relativedelta(
                months=record.duration_months
            )
        else:
            record.expiration_date = False
```

FIGURE 3 – Calcul de la date d'expiration

4.1.4 Contraintes et Validations

Des contraintes SQL et Python assurent l'intégrité des données :

```
# Contraintes SQL
_sql_constraints = [
    ('serial_number_unique', 'UNIQUE(serial_number)',
     'Le numéro de série doit être unique !'),
    ('duration_positive', 'CHECK(duration_months > 0)',
     'La durée de garantie doit être positive !'),
]
```

```
Qodo: Test this method
@api.constrains('purchase_date')
def _check_purchase_date(self):
    """Vérifie que la date d'achat n'est pas dans le futur"""
    for record in self:
        if record.purchase_date and record.purchase_date > fields.Date.today():
            raise ValidationError("La date d'achat ne peut pas être dans le futur !")
```

FIGURE 4 – Contraintes du modèle

4.2 Les Vues XML

Les vues définissent l'interface utilisateur du module.

4.2.1 Vue Liste (Tree View)

La vue liste présente un tableau récapitulatif avec code couleur :

```
        <!-- Vue Arborescence -->
        <record id="view_gestion_garantie_tree" model="ir.ui.view">
            <field name="name">gestion.garantie.tree</field>
            <field name="model">gestion.garantie</field>
            <field name="arch" type="xml">
                <list decoration-success="state=='valid'"
                      decoration-danger="state=='expired'"
                      decoration-warning="state=='claim'"
                      decoration-muted="state=='draft'">
                    <field name="name"/>
                    <field name="partner_id"/>
                    <field name="product_id"/>
                    <field name="serial_number"/>
                    <field name="purchase_date"/>
                    <field name="expiration_date"/>
                    <field name="state" widget="badge"
                           decoration-success="state=='valid'"
                           decoration-danger="state=='expired'"
                           decoration-warning="state=='claim'"
                           decoration-info="state=='draft'"/>
                </list>
            </field>
        </record>
```

FIGURE 5 – Vue liste avec décoration

4.2.2 Vue Formulaire

Le formulaire est organisé avec une barre de statut et des sections logiques :

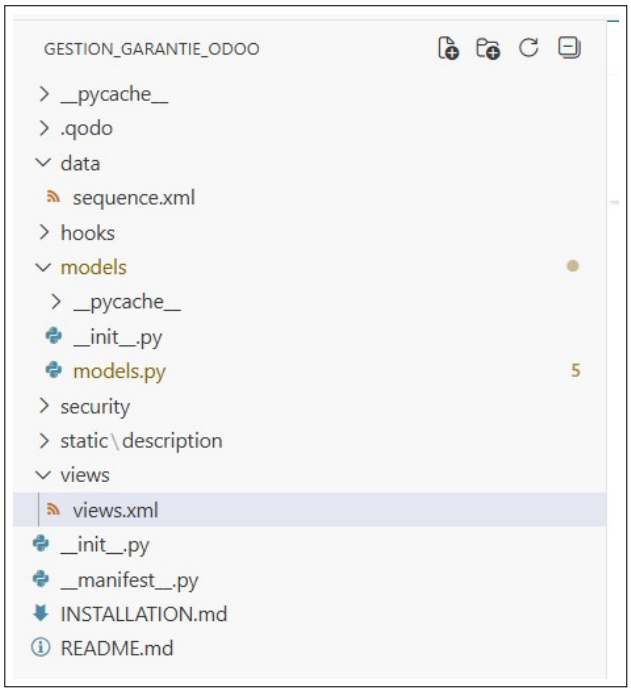
```
<odoo>
    <data>
        <record id="view_gestion_garantie_form" model="ir.ui.view">
            <field name="arch" type="xml">
                <form>
                    <header>
                        <button name="action_validate" string="Valider" type="object" ···
                                invisible="state != 'draft'"/>

                        <button name="action_create_claim" string="Créer Réclamation" ···
                                invisible="state not in ['valid', 'expired']"/>

                        <button name="action_mark_expired" string="Marquer Expirée" ···
                                invisible="state != 'valid'"/>

                        <button name="action_reset_to_draft" string="Réinitialiser" ···
                                invisible="state not in ['claim', 'expired']"/>

                        <field name="state" widget="statusbar"
                               statusbar_visible="draft,valid,expired,claim"/>
                    </header>
                    <sheet>···
                    </sheet>
                </form>
            </field>
        </record>
```

FIGURE 6 – Structure du formulaire

5 Installation et Configuration

5.1 Structure du Module

Le module `gestion_garantie_odoo` doit être placé dans le répertoire `addons` d'Odoo. La structure est la suivante :



5.2 Installation dans Odoo

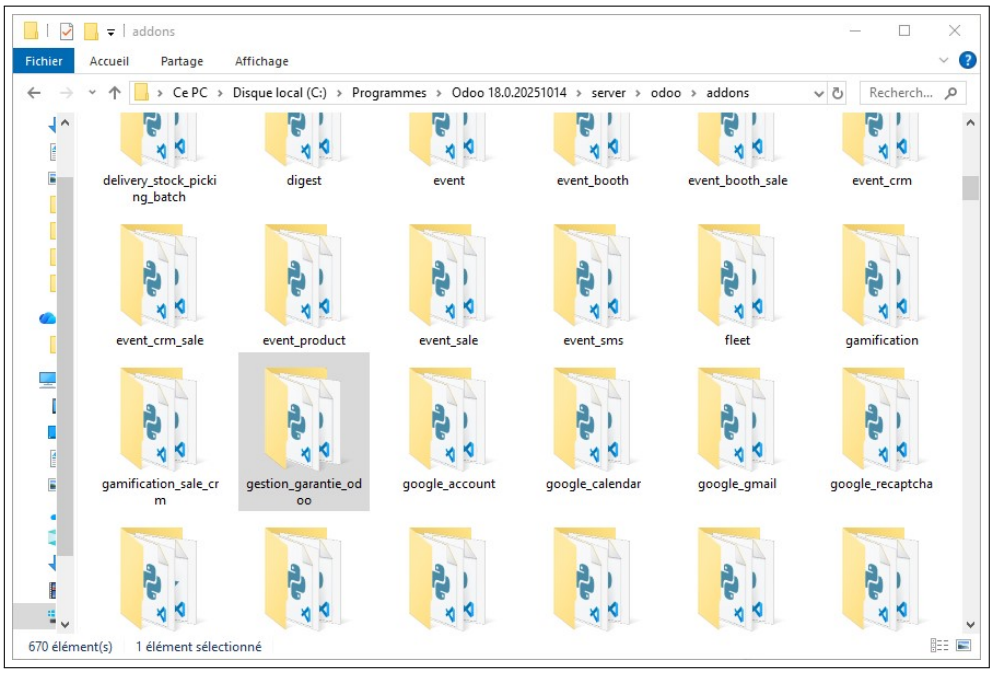


FIGURE 7 – Module présent dans le répertoire addons d'Odoo

Étapes d'installation :

1. Copier le module dans le répertoire `addons`
2. Redémarrer le serveur Odoo
3. Activer le mode développeur
4. Mettre à jour la liste des applications
5. Rechercher "Gestion des Garanties"
6. Cliquer sur "Installer"

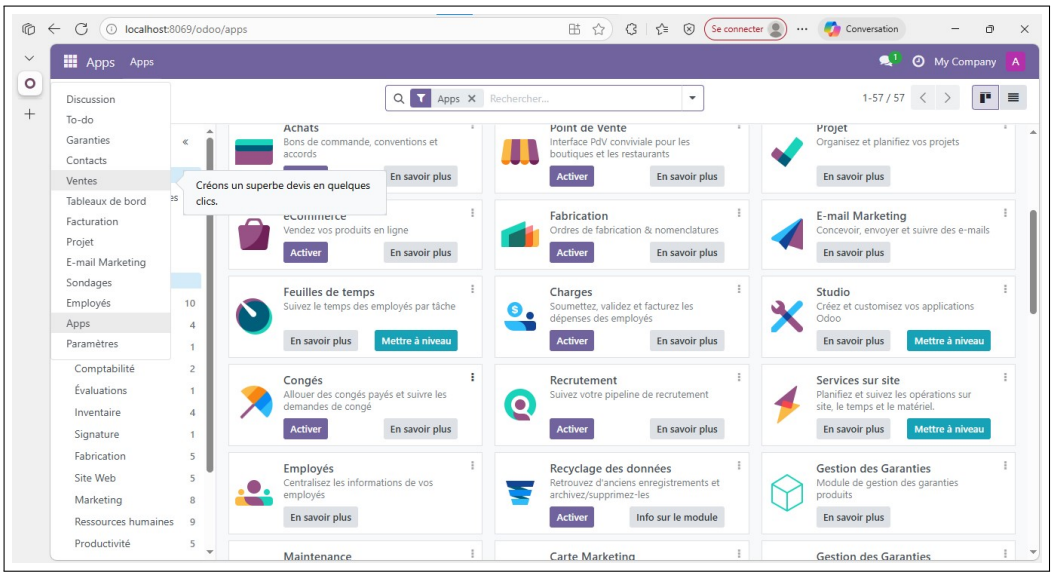


FIGURE 8 – Module visible dans la liste des applications Odoo

6 Utilisation du Module

6.1 Interface Principale

Une fois installé, le module est accessible depuis le menu principal d'Odoo.

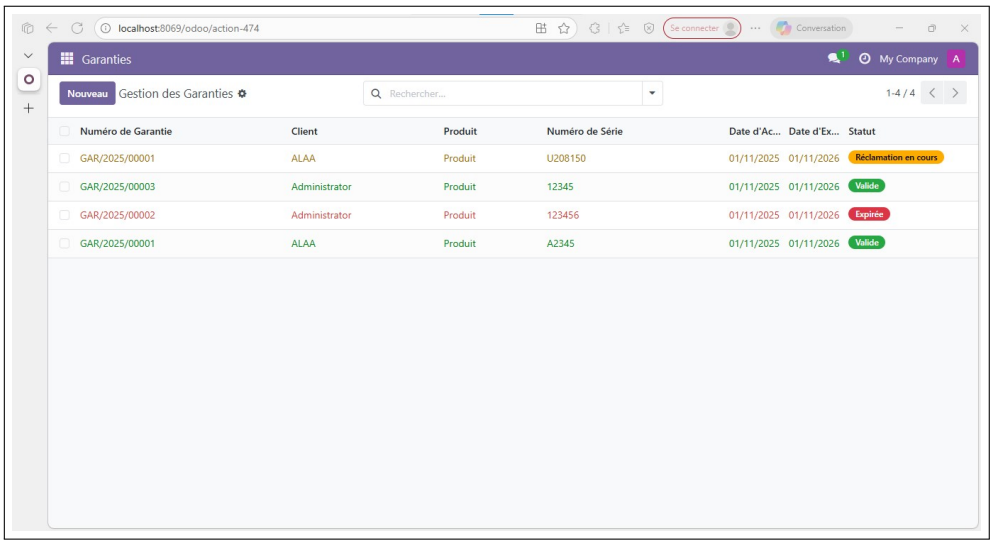


FIGURE 9 – Vue liste des garanties avec code couleur par statut

La vue liste affiche toutes les garanties avec un code couleur intuitif :

— **Vert** : Garantie valide
— **Rouge** : Garantie expirée
— **Orange** : Réclamation en cours
— **Gris** : Brouillon

6.2 Création d'une Nouvelle Garantie

6.2.1 Étape 1 : Sélection du Client

Pour créer une garantie, l'utilisateur clique sur "Nouveau" et sélectionne le client concerné.

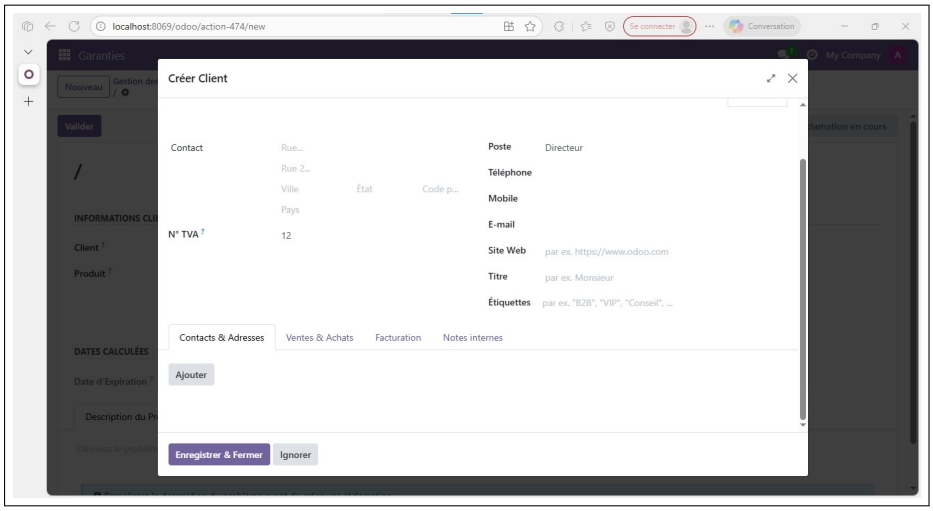


FIGURE 10 – Création d'un nouveau client via la fenêtre modale

Si le client n'existe pas, il peut être créé directement depuis cette interface.

6.2.2 Étape 2 : Sélection du Produit

De même, le produit peut être sélectionné ou créé à la volée.

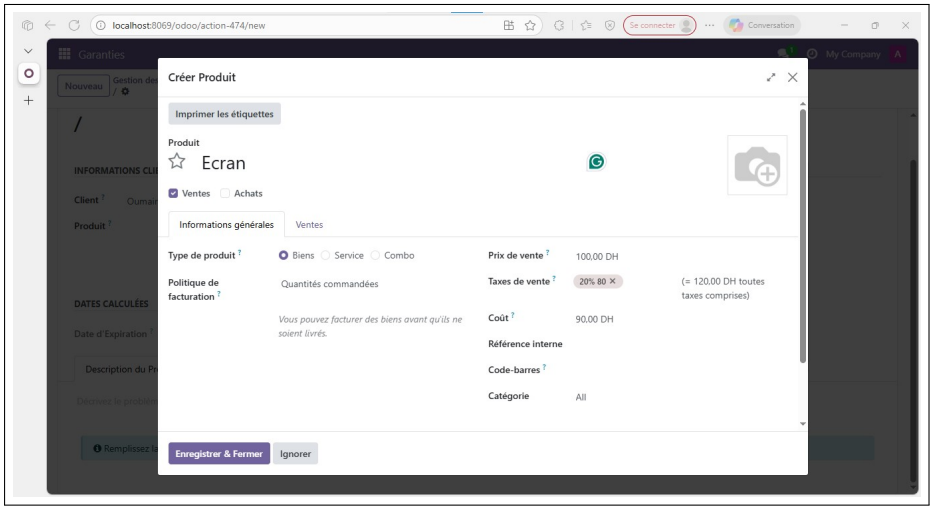


FIGURE 11 – Création d'un produit avec ses caractéristiques

6.2.3 Étape 3 : Remplissage des Informations

L'utilisateur complète les informations de la garantie :

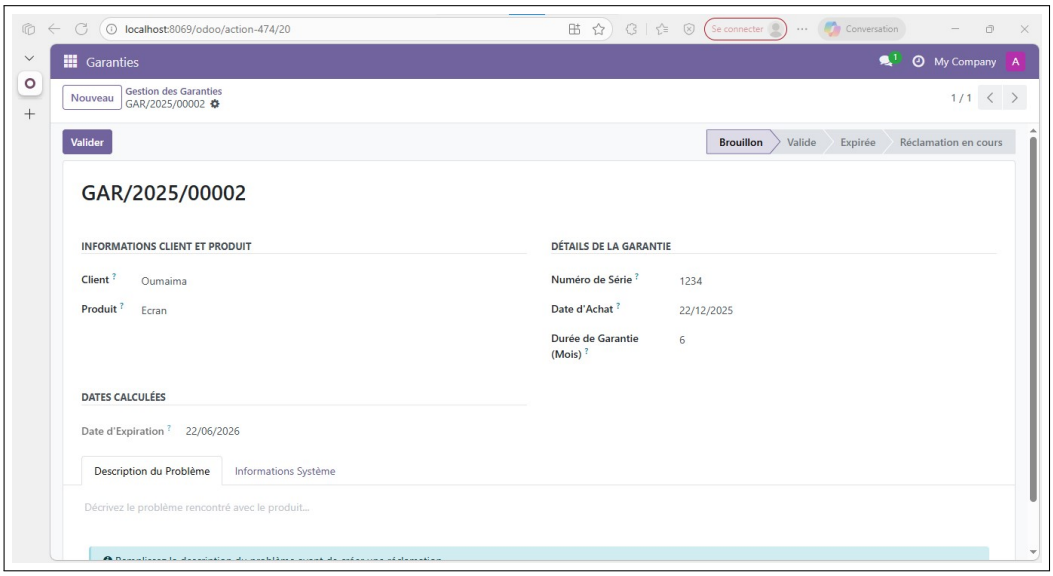


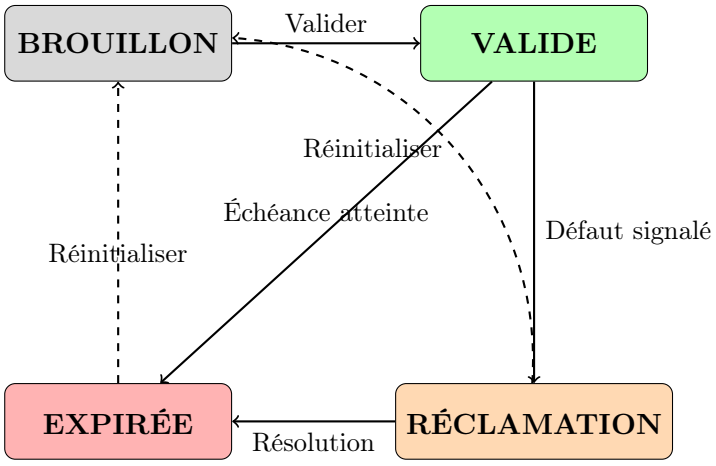
FIGURE 12 – Formulaire de garantie en statut Brouillon

Champs requis :

- Client
- Produit
- Numéro de série (unique)
- Date d'achat
- Durée de garantie en mois

La date d'expiration est **calculée automatiquement** à partir de la date d'achat et de la durée.

6.3 Cycle de Vie d'une Garantie



6.3.1 État Valide

Après validation, la garantie passe à l'état "Valide" :

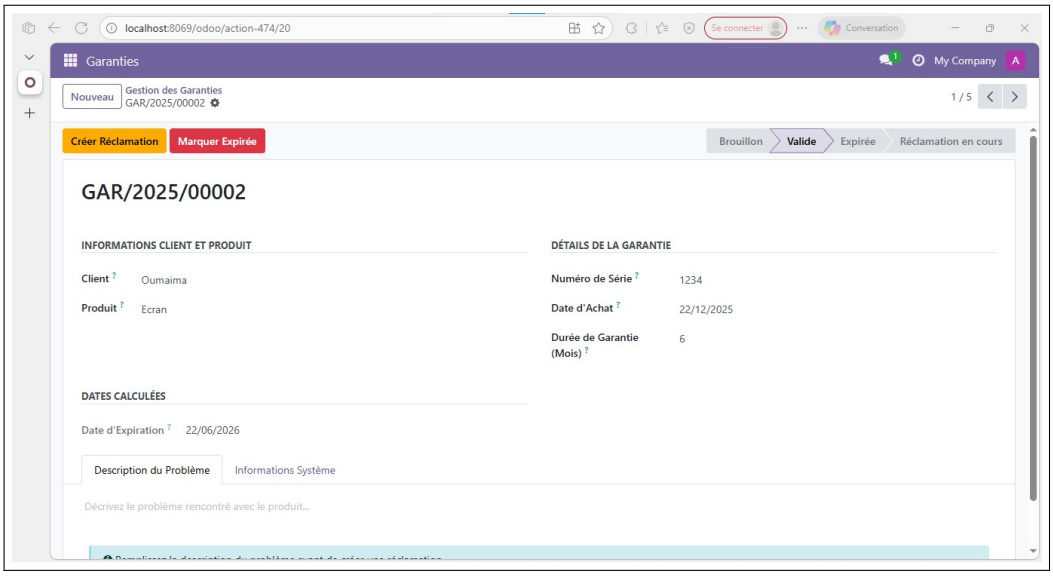


FIGURE 13 – Garantie validée avec actions disponibles

De nouveaux boutons apparaissent :

— **Créer Réclamation** : Pour signaler un défaut produit

— **Marquer Expirée** : Pour clôturer manuellement la garantie

6.3.2 Gestion des Réclamations

Lorsqu'un client signale un problème, l'utilisateur peut créer une réclamation :

1. Remplir la description du problème dans l'onglet dédié
2. Cliquer sur "Créer Réclamation"
3. Le statut passe automatiquement à "Réclamation en cours"
4. La description devient en lecture seule pour préserver l'historique

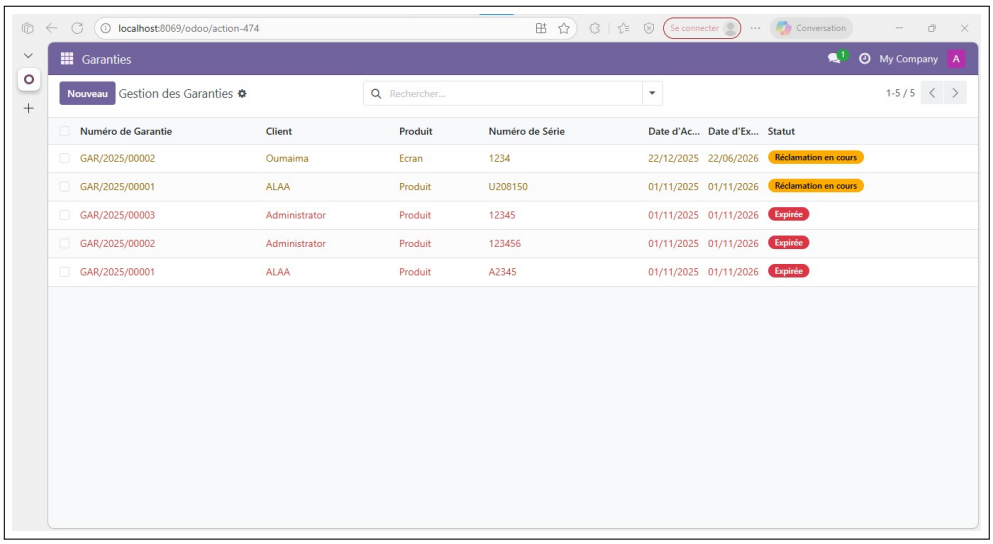


FIGURE 14 – Vue liste montrant différents statuts de garanties

6.4 Recherche et Filtrage

Le module offre des capacités de recherche avancée :

— Recherche par numéro de garantie
— Recherche par client
— Recherche par produit
— Recherche par numéro de série
— Filtres par statut (Brouillon, Valide, Expirée, Réclamation)
— Groupement par client, produit ou statut

6.5 Export des Données

Les données peuvent être exportées vers Excel pour analyse :

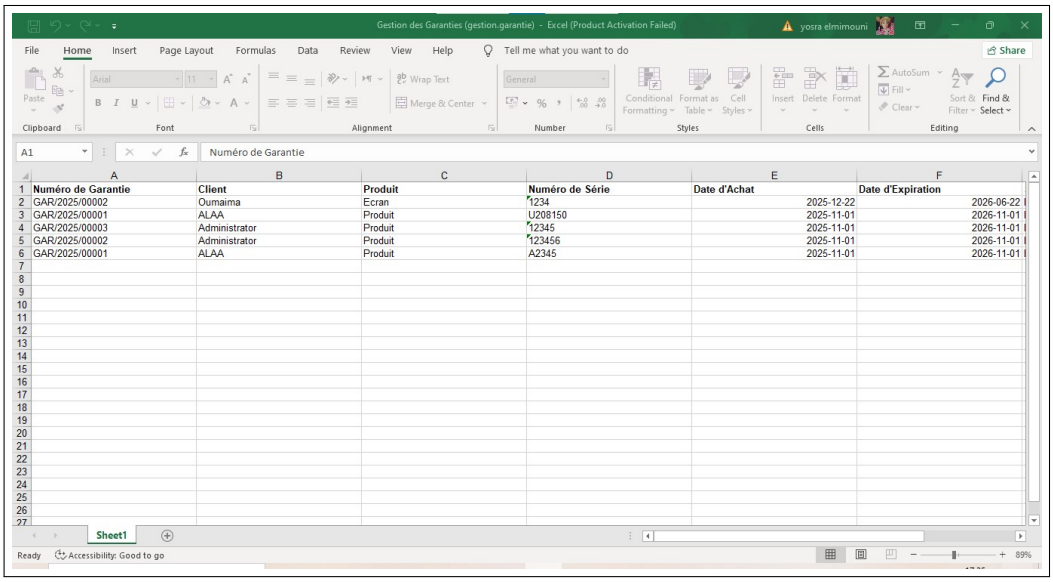


FIGURE 15 – Export des garanties vers Excel

L'export inclut tous les champs visibles :

— Numéro de garantie
— Client
— Produit
— Numéro de série
— Dates d'achat et d'expiration
— Statut

7 Fonctionnalités Techniques Avancées

7.1 Méthodes d'Action

Le modèle implémente plusieurs méthodes pour gérer le cycle de vie :

```python
    def action_validate(self):
        """Valider la garantie."""
        today = fields.Date.today()
        for record in self:
            if record.expiration_date and record.expiration_date < today:
                raise ValidationError("Impossible de valider une garantie expirée !")
            record.write({'state': 'valid'})
        return True

    Qodo: Test this method
    def action_mark_expired(self):
        """Marquer la garantie comme expirée."""
        self.write({'state': 'expired'})
        return True
```

```python
    Qodo: Test this method
    def action_create_claim(self):
        """Créer une réclamation pour la garantie."""
        self.ensure_one()
        if not self.problem_description:
            raise ValidationError(
                "Veuillez décrire le problème avant de créer une réclamation !"
            )
        self.write({'state': 'claim'})
        return True

    Qodo: Test this method
    def action_reset_to_draft(self):
        """Réinitialiser la garantie au statut brouillon."""
        self.write({'state': 'draft'})
        return True
```

FIGURE 16 – Méthodes d'action du workflow

7.2 Sécurité et Droits d'Accès

Le fichier `ir.model.access.csv` définit les permissions :

views.xml | ir.model.access.csv | sequence.xml | __init__.py hooks | __init__.py .\ | models.py 5 | __init__.py m

security > ir.model.access.csv > data

```
id,name,model_id:id,group_id:id,perm_read,perm_write,perm_create,perm_unlink
access_gestion_garantie_user,Gestion Garantie User Access,model_gestion_garantie,base.group_user,1,1,1,1
```

Cette configuration donne accès complet aux utilisateurs internes.

7.3 Intégration avec d'Autres Modules

Le module s'intègre naturellement avec :

- **Contacts** : Utilisation du modèle `res.partner`
- **Produits** : Utilisation du modèle `product.product`
- **Ventes** : Possibilité d'extension future pour création automatique

8 Avantages et Bénéfices

8.1 Gains Opérationnels

Bénéfices Mesurables

- **Réduction du temps de traitement** : gain de temps sur la gestion quotidienne
- **Élimination des erreurs** : Calculs automatisés sans risque d'erreur manuelle
- **Amélioration de la réactivité** : Accès immédiat à l'historique complet
- **Centralisation** : Une seule source de vérité pour toutes les garanties
- **Traçabilité complète** : Audit trail automatique de toutes les modifications

8.2 Satisfaction Client

Le module contribue directement à améliorer la satisfaction client :

- Réponse rapide aux demandes de statut
- Suivi rigoureux des réclamations
- Historique complet et accessible

9 Perspectives et Évolutions Futures

9.1 Fonctionnalités Planifiées

Roadmap d'Évolution

Phase 1 : Automatisation Avancée

1. **Génération automatique lors de la vente**
   - Création automatique à la confirmation de commande
   - Liaison directe avec les lignes de commande
   - Pré-remplissage des informations produit
2. **Système de notifications**
   - Alertes email avant expiration (30, 15, 7 jours)
   - Notification au client et au gestionnaire
   - Rappels automatiques pour les réclamations non résolues

Phase 2 : Portail Client

3. **Interface web dédiée**
   - Consultation en ligne du statut des garanties
   - Création de réclamations par le client
   - Upload de photos/documents justificatifs
   - Suivi en temps réel de l'avancement

Phase 3 : Analytics et Reporting

4. **Tableaux de bord**
   - Statistiques sur les réclamations par produit
   - Analyse des causes de défaillance
   - Indicateurs de performance
   - Prévisions de charge de travail
5. **Rapports personnalisés**
   - Génération de rapports PDF
   - Exports personnalisés
   - Graphiques et visualisations

9.2 Intégrations Possibles

- **Service Helpdesk** : Création automatique de tickets SAV
- **Inventaire** : Suivi des pièces de rechange utilisées
- **Comptabilité** : Gestion des coûts de garantie
- **CRM** : Enrichissement du profil client

10 Difficultés Rencontrées et Solutions

10.1 Défis Techniques

Au cours du développement, plusieurs défis ont été relevés :

Défis et Solutions

1. **Génération de la séquence**
   - *Problème* : La séquence n'existait pas au premier lancement
   - *Solution* : Méthode `_get_or_create_sequence()` qui crée la séquence si nécessaire
2. **Calcul des dates avec gestion des mois**
   - *Problème* : Différence entre les mois (28-31 jours)
   - *Solution* : Utilisation de `relativedelta` de la librairie `dateutil`
3. **Gestion de la description du problème**
   - *Problème* : Éviter la modification après création de réclamation
   - *Solution* : Champ en lecture seule quand `state == 'claim'`
4. **Validation des données**
   - *Problème* : Assurer l'intégrité des données
   - *Solution* : Contraintes SQL et validations Python (`@api.constrains`)

10.2 Améliorations Apportées

Suite aux tests, plusieurs améliorations ont été implémentées :

- Vérification avant validation (garantie non expirée)
- Obligation de description avant création de réclamation
- Code couleur intuitif dans les listes
- Messages d'aide contextuels

11 Guide de Maintenance

11.1 Mise à Jour du Module

Pour mettre à jour le module après modifications :

1. Modifier les fichiers nécessaires
2. Incrémenter la version dans `__manifest__.py`
3. Redémarrer le serveur Odoo
4. Aller dans Apps > Mise à jour de la liste des apps
5. Rechercher le module et cliquer sur "Mettre à jour"

12 Conclusion

12.1 Réalisations

Ce projet a permis de développer un module Odoo complet et fonctionnel qui répond aux besoins de gestion des garanties de l'entreprise. Les objectifs fixés ont été atteints :

- Centralisation des données de garantie
- Automatisation des calculs
- Interface utilisateur intuitive
- Traçabilité complète
- Gestion du cycle de vie
- Export des données

12.2 Compétences Acquises

Ce projet a permis de développer des compétences techniques importantes :

- Maîtrise du framework Odoo
- Programmation Python orientée objet
- Conception de modèles de données relationnels
- Développement d'interfaces XML
- Validation et sécurité des données

Ce module de gestion des garanties constitue une première étape réussie vers la digitalisation complète du service après-vente. Son architecture modulaire et évolutive permet d'envisager sereinement les développements futurs pour répondre aux besoins croissants de l'entreprise.

Fin du Rapport

Développé avec Odoo 18.0
Module ERP - 29 décembre 2025