Mise en place d'une solution de gestion des déplacements

Module Odoo 19

Ahmed Raji

15 Novembre 2025

Table des matières

1 Introduction

1.1 Contexte du projet

La gestion des déplacements professionnels constitue un enjeu majeur pour les entreprises modernes. Elle implique la coordination de multiples acteurs (employés, managers, direction administrative et financière) et nécessite un suivi rigoureux des demandes, des validations et des coûts associés.

Ce projet vise à développer un module Odoo 19 dédié à la gestion complète du cycle de vie des demandes de déplacement, depuis leur création par les employés jusqu'à leur validation finale et leur clôture administrative.

1.2 Objectifs du projet

— Automatiser le processus de demande de déplacement
— Implémenter un workflow de validation à trois niveaux (Employé → Manager → DAF)
— Calculer automatiquement les frais de mission selon des règles métier prédéfinies
— Assurer la traçabilité complète des actions et décisions
— Gérer les notifications et communications entre les acteurs
— Garantir la sécurité et les droits d'accès appropriés

1.3 Technologies utilisées

— **Odoo 19.0** — Plateforme ERP open-source
— **Python 3** — Langage de programmation pour les modèles métier
— **XML** — Format pour les vues, données et templates
— **PostgreSQL** — Base de données relationnelle
— **Docker** — Conteneurisation pour l'environnement de développement

2 Conception

2.1 Diagramme de classes

FIGURE 1 – Diagramme de classes du module gestionDeplacements

2.2 Diagramme de cas d'utilisation

FIGURE 2 – Diagramme de cas d'utilisation du système de gestion des déplacements

Le diagramme de cas d'utilisation illustre les interactions entre les trois types d'acteurs et le système. Les acteurs héritent de leurs droits de manière hiérarchique : le DAF possède tous les droits du Manager, qui lui-même possède tous les droits de l'Employé.

2.2.1 Acteurs

— **Employé** — Utilisateur de base qui crée et soumet des demandes
— **Manager** — Validateur de premier niveau (hérite des droits Employé)
— **DAF** — Direction Administrative et Financière, validateur final (hérite des droits Manager)

2.2.2 Cas d'utilisation par acteur

extbfEmployé :
— `Créer une demande` — Initialiser une nouvelle demande de déplacement
— `Soumettre une demande` — Envoyer la demande pour validation
— `Consulter ses demandes` — Visualiser l'état de ses demandes
— `Annuler une demande` — Retirer une demande soumise
extbfManager (+ droits Employé) :
— `Valider une demande` — Approuver une demande soumise par un membre de l'équipe
— `Refuser une demande` — Rejeter une demande avec motif obligatoire
extbfDAF (+ droits Manager + Employé) :
— `Traiter une demande` — Prendre en charge une demande approuvée
— `Terminer une demande` — Clôturer définitivement le processus
— `Gérer la configuration` — Administrer les villes et véhicules de service

2.3 Description des classes

2.3.1 Classe `hr.employee` (Odoo existante)

Rôle : Représente les employés de l'entreprise. Cette classe fait partie du module HR d'Odoo.
Attributs principaux :
— `name : String` — nom complet de l'employé
— `user_id : Many2one` — lien vers le compte utilisateur
— `parent_id : Many2one` — référence au manager direct
— `work_email : String` — email professionnel
Relations :
— Association 1..* avec `DemandeDeplacement` — un employé peut créer plusieurs demandes
— Auto-référence via `parent_id` pour la hiérarchie managériale

2.3.2 Classe `gestion.deplacement.demande`

Rôle : Classe centrale du système. Elle représente une demande de déplacement avec toute sa logique métier.
Attributs principaux :
— `name : String` — référence unique (ex : DEP00001)
— `employee_id : Many2one` — employé demandeur
— `manager_id : Many2one` — manager validateur (calculé automatiquement)
— `date_debut : Date` — date de début du déplacement
— `date_fin : Date` — date de fin du déplacement
— `nb_jours : Integer` — nombre de jours (calculé)

— `destination_city_id : Many2one` — ville de destination
— `is_international : Boolean` — indicateur de déplacement international (calculé)
— `mode_transport : Selection` — moyen de transport choisi
— `distance_estimee : Float` — distance en kilomètres
— `classe_voyage : Selection` — classe de voyage pour l'avion (calculé)
— `vehicule_id : Many2one` — véhicule de service si applicable
— `montant_frais : Monetary` — montant des frais estimés (calculé)
— `mission_objet : Text` — description de la mission
— `ordre_mission_file : Binary` — fichier PDF de l'ordre de mission
— `state : Selection` — état du workflow
— `motif_refus : Text` — raison du refus si applicable

États du workflow :
— `brouillon` — Demande en création par l'employé
— `soumis` — Demande soumise au manager
— `approuve` — Demande approuvée par le manager
— `en_cours` — Demande prise en charge par la DAF
— `termine` — Demande terminée et archivée
— `refuse` — Demande refusée par le manager
— `annule` — Demande annulée

Héritage Odoo :
— `mail.thread` — Active le Chatter pour les messages et historique
— `mail.activity.mixin` — Active le système d'activités et notifications

Relations :
— Many2one avec `hr.employee` (employé demandeur)
— Many2one avec `hr.employee` (manager validateur)
— Many2one avec `gestion.deplacement.ville` (destination)
— Many2one avec `gestion.deplacement.service.vehicule` (optionnel)

2.3.3 Classe `gestion.deplacement.ville`

Rôle : Représente les villes de destination possibles.

Attributs :
— `name : String` — nom de la ville
— `country_id : Many2one` — pays de la ville
— `code_postal : String` — code postal
— `active : Boolean` — indicateur actif/archivé

Contraintes :
— Contrainte unique sur (name, country_id) — évite les doublons

Relations :
— Many2one avec `res.country` (classe Odoo existante)
— One2many inverse avec `DemandeDeplacement`

2.3.4 Classe `gestion.deplacement.service.vehicule`

Rôle : Représente les véhicules de service de l'entreprise.

Attributs :
— `name : String` — nom du véhicule
— `immatriculation : String` — numéro d'immatriculation

— `marque : String` — marque du véhicule
— `modele : String` — modèle du véhicule
— `company_id : Many2one` — société propriétaire
— `active : Boolean` — indicateur actif/archivé

Relations :

— One2many inverse avec `DemandeDeplacement`

2.3.5 Classe `gestion.deplacement.demande.refus.wizard`

Rôle : Modèle transitoire (wizard) pour la saisie du motif de refus.
Type : TransientModel — données temporaires non persistées
Attributs :

— `motif_refus : Text` — raison du refus (obligatoire)
— `demande_id : Many2one` — demande concernée

Méthode :

— `action_confirm_refus()` — Applique le refus avec le motif saisi

2.3.6 Énumération `mode_transport`

Rôle : Définit les modes de transport possibles.
Valeurs :

— `train` — Train
— `autocar` — Autocar
— `avion` — Avion
— `vehicule_service` — Véhicule de service

2.3.7 Énumération `classe_voyage`

Rôle : Définit les classes de voyage pour l'avion (calculé automatiquement).
Valeurs :

— `economique` — Classe économique (distance < 6000 km)
— `business` — Classe business (distance ≥ 6000 km)

2.4 Relations clés résumées

Relation	Type	Description
Employee → DemandeDeplacement	1..*	Un employé peut créer plusieurs demandes
Manager → DemandeDeplacement	1..*	Un manager valide plusieurs demandes
DemandeDeplacement → Ville	1..1	Chaque demande a une destination
Ville → Pays	1..1	Chaque ville appartient à un pays
DemandeDeplacement → Vehicule	0..1	Optionnel si mode = véhicule de service
DemandeDeplacement → ModeTransport	1..1	Mode de transport obligatoire

TABLE 1 – Principales relations entre les classes

2.5 Distinction entre classes Odoo et personnalisées

Type	Description
hr.employee	Classe Odoo existante du module HR
res.country	Classe Odoo existante pour les pays
res.currency	Classe Odoo existante pour les devises
res.company	Classe Odoo existante pour les sociétés
mail.thread	Mixin Odoo pour le Chatter
mail.activity.mixin	Mixin Odoo pour les activités
gestion.deplacement.demande	Classe personnalisée créée
gestion.deplacement.ville	Classe personnalisée créée
gestion.deplacement.service.vehicule	Classe personnalisée créée
gestion.deplacement.demande.refus.wizard	Classe personnalisée créée

TABLE 2 – Distinction entre classes Odoo existantes et personnalisées

2.6 Règles métier automatisées

2.6.1 Calcul du nombre de jours

Le nombre de jours est calculé automatiquement à partir des dates de début et de fin :

```
@api.depends('date_debut', 'date_fin')
def _compute_nb_jours(self):
    for record in self:
        if record.date_debut and record.date_fin:
            delta = record.date_fin - record.date_debut
```

```
            record.nb_jours = delta.days + 1 if delta.days >= 0
                else 1
```

Listing 1 – Calcul automatique du nombre de jours

2.6.2 Détermination du caractère international

Le système détermine automatiquement si le déplacement est international en comparant le pays de destination avec le pays de la société :

```
@api.depends('destination_city_id', 'destination_city_id.country_id
    ')
def _compute_is_international(self):
    company_country = self.env.company.country_id
    for record in self:
        if record.destination_city_id and record.
           destination_city_id.country_id and company_country:
            record.is_international = (record.destination_city_id.
               country_id.id != company_country.id)
```

Listing 2 – Détection automatique des déplacements internationaux

2.6.3 Calcul des frais de mission

Le montant des frais est calculé selon des barèmes différents pour les déplacements nationaux et internationaux :

- **National :** 700 MAD/jour
- **International :** 1500 MAD/jour

```
@api.depends('nb_jours', 'is_international')
def _compute_montant_frais(self):
    for record in self:
        if record.is_international:
            record.montant_frais = record.nb_jours * 1500.0
        else:
            record.montant_frais = record.nb_jours * 700.0
```

Listing 3 – Calcul automatique des frais

2.6.4 Calcul de la classe de voyage

Pour les déplacements en avion, la classe est déterminée automatiquement selon la distance :

- **Distance $<$ 6000 km :** Classe économique
- **Distance $\geq$ 6000 km :** Classe business

```
@api.depends('distance_estimee', 'mode_transport')
def _compute_classe_voyage(self):
    for record in self:
        if record.mode_transport == 'avion':
            if record.distance_estimee > 6000:
                record.classe_voyage = 'business'
```

```
7                else:
8                    record.classe_voyage = 'economique'
```

Listing 4 – Calcul automatique de la classe avion

2.7 Contraintes de validation

Le système implémente plusieurs contraintes pour garantir la cohérence des données :

Contrainte	**Description**
Cohérence des dates	La date de fin doit être $\geq$ date de début
Distance positive	La distance doit être > 0 km
Règle avion	L'avion nécessite une distance ≥ 500 km
Véhicule obligatoire	Si mode = véhicule_service, un véhicule doit être sélectionné
Ordre de mission	Obligatoire avant soumission
Employé unique	Un employé ne peut créer des demandes que pour lui-même

TABLE 3 – Contraintes de validation métier

3 Sécurité et Droits d'Accès

3.1 Groupes de sécurité

Le module définit trois niveaux de sécurité hiérarchiques :

3.1.1 Groupe 1 : Employé (group_deplacement_employee)

Droits :
- Créer ses propres demandes de déplacement
- Modifier ses demandes en état `brouillon`
- Soumettre ses demandes pour validation
- Consulter uniquement ses propres demandes
- Annuler ses demandes
- Remettre en brouillon les demandes refusées

3.1.2 Groupe 2 : Manager (group_deplacement_manager)

Droits (en plus de ceux de l'employé) :
- Consulter les demandes de son équipe
- Approuver ou refuser les demandes soumises
- Accéder au wizard de refus pour saisir un motif

3.1.3 Groupe 3 : DAF (group_deplacement_daf)

Droits (en plus de ceux du manager) :

— Consulter toutes les demandes de l'entreprise
— Prendre en charge les demandes approuvées
— Marquer les demandes comme terminées
— Gérer la configuration (villes, véhicules)
— Accès complet en lecture/écriture/suppression

3.2 Record Rules

Groupe	Domaine	Accès
Employé	`[('employee_id.user_id', '=', user.id)]`	Ses demandes uniquement
Manager	`[('manager_id.user_id', '=', user.id)]`	Demandes de son équipe
DAF	`[(1, '=', 1)]`	Toutes les demandes

TABLE 4 – Règles de sécurité par groupe

4 Workflow et Notifications

4.1 Diagramme de flux du workflow

Le processus de validation suit un workflow séquentiel à trois niveaux :

```
[Brouillon] --Soumettre--> [Soumis] --Approuver--> [Approuvé]
                              |
                           Refuser
                              |
                              v
                           [Refusé] --Remettre en brouillon--> [Brouillon]

[Approuvé] --Prendre en charge--> [En cours] --Terminer--> [Terminé]
```

4.2 Actions et transitions

Action	État initial	État final	Acteur
Soumettre	brouillon	soumis	Employé
Approuver	soumis	approuve	Manager
Refuser	soumis/approuve	refuse	Manager
Prendre en charge	approuve	en_cours	DAF
Terminer	en_cours	termine	DAF
Annuler	*	annule	Employé/DAF
Remettre en brouillon	refuse	brouillon	Employé

TABLE 5 – Actions du workflow

4.3 Système de notifications

Le module utilise trois mécanismes de notification complémentaires :

4.3.1 Activités (horloge)

Rôle : Notifications de tâches pour les actions requises
Utilisation :
- Notification au manager lors de la soumission d'une demande
- Notification à tous les DAF lors de l'approbation d'une demande
- Affichage dans l'icône (horloge) avec compteur rouge

```
self.activity_schedule(
    'mail.mail_activity_data_todo',
    user_id=self.manager_id.user_id.id,
    summary=f"Demande de deplacement a valider: {self.name}",
    note=f"L'employe {self.employee_id.name} a soumis une demande."
)
```

Listing 5 – Création d'une activité pour le manager

4.3.2 Chatter (Messages)

Rôle : Historique complet des actions et communications
Utilisation :
- Enregistrement de chaque transition d'état
- Messages système avec émojis pour meilleure lisibilité
- Visible par tous les followers de la demande

```
self.message_post(
    body="Demande soumise pour validation par le manager",
    subtype_xmlid='mail.mt_comment'
)
```

Listing 6 – Post de message dans le Chatter

4.3.3 Emails

Rôle : Notifications externes par email (optionnel)

Templates disponibles :

— `mail_template_demande_submitted` — Notification de soumission au manager
— `mail_template_demande_approved` — Notification d'approbation à l'employé
— `mail_template_demande_approved_daf` — Notification de traitement aux DAF
— `mail_template_demande_refused` — Notification de refus à l'employé

Gestion des erreurs :

```
try:
    template.send_mail(self.id, force_send=True)
except Exception as e:
    self.message_post(body=f"Erreur␣lors␣de␣l'envoi␣de␣l'email:␣{
       str(e)}")
```

Listing 7 – Envoi sécurisé d'emails avec gestion d'erreurs

4.4 Système de followers

Le système ajoute automatiquement les parties prenantes comme followers :
— Le **manager** est ajouté lors de la soumission
— Tous les **DAF** sont ajoutés lors de l'approbation
— Les followers reçoivent les mises à jour du Chatter

```
self.message_subscribe(partner_ids=[self.manager_id.user_id.
   partner_id.id])
```

Listing 8 – Ajout automatique de followers

5 Réalisation

5.1 Structure du module

```
gestionDeplacements/
|-- __init__.py
|-- __manifest__.py
|-- models/
|   |-- __init__.py
|   |-- demande_deplacement.py
|   |-- ville.py
|   `-- service_vehicule.py
|-- views/
|   |-- demande_deplacement_view.xml
|   |-- ville_view.xml
|   |-- service_vehicule_view.xml
```

```
|   `-- menu.xml
|-- wizard/
|   |-- __init__.py
|   |-- demande_refus_wizard.py
|   `-- demande_refus_wizard_view.xml
|-- security/
|   |-- deplacement_security.xml
|   `-- ir.model.access.csv
|-- data/
|   |-- sequence.xml
|   |-- ville_data.xml
|   |-- service_vehicule_data.xml
|   `-- email_templates.xml
`-- static/
    `-- description/
```

5.2 Interface utilisateur : État Brouillon

FIGURE 3 – Interface de création d'une nouvelle demande (état Brouillon)

L'interface de création permet à l'employé de saisir toutes les informations nécessaires :
— **Informations générales :** Employé (auto-rempli), Manager (calculé), Dates
— **Destination :** Ville de destination (avec recherche), Distance
— **Transport :** Mode de transport, Véhicule (si applicable)
— **Ordre de mission :** Upload du fichier PDF obligatoire
— **Mission :** Description de l'objet de la mission

Les champs calculés (Nombre de jours, Montant estimé, Classe voyage) se mettent à jour automatiquement.

5.3 État Soumis

FIGURE 4 – Demande soumise pour validation

Après soumission :
— L'état passe à **Soumis**
— Le formulaire devient en lecture seule
— Le manager reçoit une activité de notification
— Un email est envoyé au manager (si SMTP configuré)
— L'employé peut annuler sa demande

5.4 Validation par le Manager

FIGURE 5 – Interface du manager avec options de validation

Le manager dispose de deux options principales :

— **Valider** — Approuve la demande et l'envoie à la DAF
— **Refuser** — Ouvre un wizard pour saisir le motif de refus

Note : L'option **Remettre en brouillon** n'apparaît que pour les demandes déjà refusées, permettant à l'employé de corriger et resoumettre sa demande.

5.5 Wizard de refus

FIGURE 6 – Popup de saisie du motif de refus

Le wizard de refus garantit qu'un motif est toujours fourni lors d'un refus :
— Champ texte obligatoire pour le motif
— Bouton "Confirmer le refus" pour valider
— Bouton "Annuler" pour fermer sans action
— Le motif est ensuite visible dans la demande et envoyé par email

5.6 État Approuvé

FIGURE 7 – Demande approuvée par le manager

Après approbation :
— L'état passe à **Approuvé**
— Tous les DAF reçoivent une activité de notification
— L'employé reçoit un email de confirmation
— Les DAF reçoivent un email détaillé pour traitement
— Les DAF peuvent prendre en charge la demande

5.7 Prise en charge par la DAF

FIGURE 8 – Interface DAF pour le traitement de la demande

La DAF dispose de trois actions :
— **Prendre en charge** — Passe l'état à "En cours" (achat de billets, réservations)
— **Refuser** — Permet de refuser la demande avec motif
— **Terminer** — Clôture définitive de la demande

5.8 État Terminé

FIGURE 9 – Demande terminée et archivée

L'état **Terminé** représente la clôture complète du processus :
— Formulaire en lecture seule
— Aucun bouton d'action disponible
— Historique complet conservé
— Demande archivée pour consultation future

5.9 Suivi du workflow (Chatter)

FIGURE 10 – Historique complet des actions dans le Chatter (bas du formulaire)

Le **Chatter**, situé en bas de chaque formulaire de demande, enregistre automatiquement l'historique complet de toutes les actions effectuées. Comme illustré dans la figure **??**, chaque événement du workflow est tracé avec précision :

- **Date et heure** — Horodatage précis de chaque action (Nov 15, 2 :55 AM / 3 :00 AM)
- **Subject** — Description de l'événement :
  - "Demande DEP00001 - À valider" — Message d'activité pour le manager
  - "[Déplacement] Demande $object.name} soumise" — Email de notification envoyé
  - "[Déplacement] Demande $object.name} approuvée" — Confirmation d'approbation
  - "[Déplacement] Demande $object.name} à traiter" — Notification à la DAF
- **Author** — Utilisateur ayant déclenché l'action (Sophie Dubois, Marie Martin)
- **Related Document** — Lien vers la demande concernée (DEP00001)

Cette section du Chatter combine plusieurs types de messages :

- **Messages système** — Créés automatiquement lors des transitions d'état
- **Emails envoyés** — Notifications aux managers et DAF
- **Activités** — Tâches assignées aux validateurs
- **Notes** — Commentaires manuels des utilisateurs (si ajoutés)

Cette traçabilité automatique offre :

- **Audit complet** — Historique immuable chronologique de toutes les actions
- **Conformité** — Preuve de validation à chaque étape du processus
- **Transparence** — Visibilité sur qui a fait quoi et quand
- **Responsabilité** — Attribution claire des décisions et validations

Tous les followers de la demande (manager, DAF, employé) peuvent consulter cet historique pour suivre l'évolution de la demande en temps réel.

5.10 Vues additionnelles

Le module propose 7 types de vues différentes :

5.10.1 Vue Liste

- Affichage tabulaire de toutes les demandes
- Tri et filtres dynamiques
- Totaux automatiques des montants
- Codes couleur selon l'état

5.10.2 Vue Kanban

- Affichage en cartes groupées par état
- Glisser-déposer pour changer d'état (si droits suffisants)
- Informations synthétiques sur chaque carte

5.10.3 Vue Calendrier

— Visualisation des déplacements dans le temps
— Groupement par employé (code couleur)
— Navigation mensuelle/hebdomadaire

5.10.4 Vue Pivot

— Tableau croisé dynamique
— Analyse multidimensionnelle (employé × état × montant)
— Export Excel/CSV

5.10.5 Vue Graph

— Graphiques en barres, courbes, secteurs
— Statistiques par état, employé, période
— Visualisation des tendances

6 Données de démonstration

6.1 Villes préchargées

Le module inclut des données de démonstration pour le Maroc :

Ville	**Pays**	**Code Postal**
Casablanca	Maroc	20000
Rabat	Maroc	10000
Marrakech	Maroc	40000
Fès	Maroc	30000
Tanger	Maroc	90000
Agadir	Maroc	80000

TABLE 6 – Villes préchargées

6.2 Véhicules de service

Nom	**Marque**	**Modèle**	**Immatriculation**
Peugeot 208	Peugeot	208	ABC-123-MA
Renault Clio	Renault	Clio	XYZ-456-MA
Dacia Logan	Dacia	Logan	DEF-789-MA
Toyota Corolla	Toyota	Corolla	GHI-012-MA
Volkswagen Golf	Volkswagen	Golf	JKL-345-MA

TABLE 7 – Véhicules de service préchargés

7 Déploiement avec Docker

7.1 Configuration Docker Compose

Le projet utilise Docker pour faciliter le déploiement et garantir la portabilité :

```yaml
version: "3.8"

services:
  odoo:
    image: odoo:19.0
    container_name: odoo-web
    ports:
      - "8069:8069"
    environment:
      - HOST=host.docker.internal
      - USER=odoo
      - PASSWORD=odoo
    volumes:
      - odoo-web-data:/var/lib/odoo
      - ./addons:/mnt/extra-addons

volumes:
  odoo-web-data:
```

Listing 9 – docker-compose.yml

7.2 Installation et démarrage

```
# 1. Démarrer les conteneurs
docker-compose up -d

# 2. Accéder à Odoo
http://localhost:8069

# 3. Créer la base de données
- Database: odoo19
- Email: admin@example.com
- Password: admin

# 4. Installer le module
Apps → Rechercher "Gestion des Déplacements" → Installer

# 5. Redémarrer si nécessaire
docker restart odoo-web
```

8 Tests et Validation

8.1 Scénarios de test

8.1.1 Test 1 : Création et soumission (Employé)

1. Se connecter en tant qu'employé
2. Créer une nouvelle demande
3. Vérifier le calcul automatique du manager
4. Remplir tous les champs obligatoires
5. Vérifier le calcul du nombre de jours
6. Vérifier le calcul du montant (national vs international)
7. Joindre un ordre de mission PDF
8. Soumettre la demande
9. Vérifier que l'état passe à "Soumis"
10. Vérifier la création de l'activité pour le manager

8.1.2 Test 2 : Validation (Manager)

1. Se connecter en tant que manager
2. Vérifier la notification d'activité
3. Ouvrir la demande soumise
4. Tester l'approbation :
   — Cliquer sur "Valider"
   — Vérifier le passage à l'état "Approuvé"
   — Vérifier la création d'activités pour les DAF
   — Vérifier l'envoi des emails
5. Tester le refus (sur une autre demande) :
   — Cliquer sur "Refuser"
   — Vérifier l'ouverture du wizard
   — Saisir un motif obligatoire
   — Confirmer le refus
   — Vérifier le passage à l'état "Refusé"
   — Vérifier l'affichage du motif

8.1.3 Test 3 : Traitement (DAF)

1. Se connecter en tant que DAF
2. Vérifier la notification d'activité
3. Ouvrir une demande approuvée
4. Cliquer sur "Prendre en charge"
5. Vérifier le passage à "En cours"
6. Cliquer sur "Terminer"
7. Vérifier le passage à "Terminé"
8. Vérifier que tous les boutons disparaissent

8.1.4 Test 4 : Contraintes de validation

1. Tester date_fin < date_debut → Erreur
2. Tester distance = 0 → Erreur
3. Tester avion avec distance < 500 km → Erreur
4. Tester mode = véhicule_service sans véhicule → Erreur
5. Tester soumission sans ordre de mission → Erreur

8.2 Résultats des tests

Test	**Résultat**	**Remarques**
Calcul automatique du manager	OK	Basé sur parent_id
Calcul du nombre de jours	OK	Inclusif (fin - début + 1)
Détection international	OK	Compare country_id
Calcul des frais	OK	700 DH (nat) / 1500 DH (int)
Calcul classe avion	OK	Business si > 6000 km
Notifications activités	OK	Affichage dans horloge
Messages Chatter	OK	Historique complet
Emails	Optionnel	Nécessite SMTP
Wizard refus	OK	Motif obligatoire
Contraintes validation	OK	Toutes les règles OK
Droits d'accès	OK	Record rules OK

TABLE 8 – Résultats des tests fonctionnels

9 Points forts et limitations

9.1 Points forts

— **Architecture modulaire** — Code bien organisé et maintenable
— **Workflow robuste** — Transitions d'état validées et sécurisées
— **Calculs automatiques** — Réduction des erreurs de saisie
— **Notifications multicouches** — Activités + Chatter + Emails
— **Sécurité granulaire** — Record rules et groupes bien définis
— **Traçabilité complète** — Historique immuable des actions
— **Gestion d'erreurs** — Emails non bloquants avec try-except
— **Interface intuitive** — 7 types de vues différentes
— **Données de démo** — Facilite les tests et la formation
— **Dockerisation** — Déploiement simplifié et portable

9.2 Limitations et améliorations possibles

— **Calcul des frais simplifié** — Barèmes fixes non paramétrables
— **Pas de gestion de disponibilité** — Véhicules sans calendrier
— **Emails optionnels** — Nécessite configuration SMTP externe
— **Classe voyage non affichée dans emails** — Champ calculé non utilisé
— **Search view commentée** — Filtres et groupements désactivés
— **Pas de validation budgétaire** — Aucune limite de dépense
— **Pas de rapports PDF** — Génération d'ordres de mission manuelle

9.3 Évolutions futures envisageables

— **Table de barèmes configurables** — Par pays, catégorie employé, etc.
— **Calendrier de disponibilité des véhicules** — Éviter les conflits
— **Intégration comptable** — Liaison avec module Accounting
— **Génération automatique de PDF** — Ordres de mission et rapports
— **Validation budgétaire** — Contrôle des enveloppes par département
— **Dashboard analytique** — KPI et statistiques en temps réel
— **API REST** — Intégration avec systèmes externes
— **Application mobile** — Soumission de demandes depuis smartphone

10 Conclusion

10.1 Bilan du projet

Ce projet a permis de développer un module Odoo 19 complet et fonctionnel pour la gestion des déplacements professionnels. L'objectif principal — automatiser le workflow de validation à trois niveaux (Employé → Manager → DAF) — a été atteint avec succès.

Le module répond aux exigences suivantes :
— Création et soumission de demandes par les employés
— Validation hiérarchique avec motif de refus obligatoire
— Calculs automatiques (jours, frais, classe voyage)
— Notifications multicouches (activités, chatter, emails)
— Sécurité granulaire avec record rules
— Traçabilité complète via Chatter
— Interface utilisateur intuitive avec 7 vues

10.2 Compétences acquises

Ce projet a permis d'approfondir les compétences suivantes :
— Développement Odoo (modèles, vues, workflows, sécurité)
— Python pour la logique métier et les calculs automatiques
— XML pour les vues et les données
— Système de notifications Odoo (activités, chatter, followers)
— Gestion de la sécurité et des droits d'accès
— Dockerisation d'applications ERP
— Conception de workflows d'approbation

10.3 Perspectives

Le module constitue une base solide pour la gestion des déplacements et peut être enrichi progressivement selon les besoins de l'entreprise. Les évolutions futures pourront porter sur l'intégration comptable, la génération automatique de documents PDF, ou encore le développement d'une application mobile pour les employés en déplacement.

Ce projet démontre la puissance et la flexibilité d'Odoo pour développer rapidement des solutions métier sur mesure, tout en bénéficiant de l'écosystème riche du framework (mail, HR, notifications, sécurité).