Université Sultan Moulay Slimane
École Nationale des Sciences Appliquées
-Khouribga-

Rapport du TP

Module : ERP

SEMESTRE 5

Filière : Génie Informatique

Développement d'un Module ERP

Réalisé par :

Ouadie	Anouar	Abdessamad	Ayoub
RACHCHAD	**RACHDI**	**SABIR**	**OULAHRAOUI**

Sous la direction de :

Pr. LAMGHARI Nidal

Année universitaire 2025/2026

Table des matières

Table des figures

Liste des tableaux

Chapitre 1

Introduction Générale

La gestion hôtelière représente un domaine hautement compétitif et complexe, où la coordination efficace de multiples processus métiers est essentielle pour assurer la rentabilité et la satisfaction client. Dans un environnement où les attentes des clients évoluent rapidement vers une personnalisation accrue et une efficacité opérationnelle, les hôtels font face à des défis tels que la gestion des réservations en temps réel, le suivi des disponibilités des chambres, la facturation dynamique, la gestion des services additionnels et la génération de rapports analytiques. Une mauvaise synchronisation entre ces éléments peut entraîner des pertes financières significatives, des conflits de réservation (surbooking), une dégradation de l'expérience client et une inefficacité opérationnelle globale.

Dans ce contexte, les systèmes émergent comme des solutions incontournables pour intégrer et automatiser ces processus. , en tant qu' open source modulaire et extensible, offre un cadre idéal pour le développement de solutions métiers personnalisées adaptées aux besoins spécifiques d'un hôtel. Ce projet vise précisément la conception, le développement et l'implémentation d'un module dédié à la gestion hôtelière, baptisé **Hotel Management**, au sein de la version 17 d'. Ce module reproduit les fonctionnalités essentielles d'un professionnel, tout en s'intégrant harmonieusement à l'écosystème pour une gestion unifiée.

Le présent rapport est structuré comme suit : nous débuterons par une présentation détaillée d' et une justification de son choix. Ensuite, nous analyserons le domaine fonctionnel et métier de la gestion hôtelière, en identifiant les problématiques clés. Une cartographie logicielle et architecturale sera exposée, suivie d'une conception UML approfondie. L'implémentation technique, incluant la logique métier et les codes sources, sera décrite en détail. Des chapitres seront consacrés à la sécurité, à l'interface utilisateur avec captures d'écran, à une comparaison avec les solutions existantes, aux tests et résultats, et enfin à une conclusion avec perspectives d'évolution.

Ce projet non seulement démontre la puissance d' pour des applications industrielles, mais illustre également comment un module personnalisé peut résoudre des problèmes concrets dans le secteur hôtelier, en favorisant l'innovation et l'efficacité.

1.1 Contexte Historique de la Gestion Hôtelière

Historiquement, la gestion hôtelière reposait sur des outils manuels comme des registres papier, ce qui limitait la scalabilité et augmentait les erreurs humaines. Avec l'avènement des technologies numériques dans les années 1980, les premiers PMS ont émergé, offrant une automatisation basique. Aujourd'hui, avec l'essor du cloud computing et des ERP intégrés, des solutions comme permettent une gestion holistique, intégrant CRM, comptabilité et analytics.

1.2 Objectifs Spécifiques du Projet

Les objectifs incluent :

— Automatiser les workflows de réservation pour éviter les surbookings.

— Intégrer des calculs dynamiques pour les coûts et totaux.

— Fournir des rapports personnalisables via des wizards.

— Assurer une interface utilisateur intuitive et sécurisée.

1.3 Enjeux Économiques et Sociétaux

Dans un marché hôtelier estimé à plusieurs billions de dollars, l'adoption d'ERP comme peut réduire les coûts opérationnels de 20-30% tout en améliorant la satisfaction client, contribuant à une économie plus durable.

Chapitre 2

Présentation d'Odoo et Justification du Choix

est un open source complet, développé en Python et utilisant PostgreSQL comme base de données. Son architecture modulaire permet d'ajouter des fonctionnalités via des modules indépendants, chacun gérant un aspect spécifique du business.

2.1 Historique et Évolution d'Odoo

Fondé en 2005 sous le nom OpenERP, a évolué vers une plateforme tout-en-un, intégrant plus de 10 000 modules communautaires. La version 17 introduit des améliorations en termes de performance, avec un ORM optimisé et une UI plus responsive.

2.2 Composants Clés d'Odoo

— **ORM** : Permet de définir des modèles Python qui mappent directement aux tables SQL, avec support pour les relations complexes (Many2one, One2many).

— **Vues XML** : Définition declarative de l'UI, incluant forms, trees, kanbans et statusbars.

— **Workflows** : Gestion des états via des boutons et des méthodes Python.

— **Reporting** : QWeb pour PDF/HTML, avec wizards pour filtres dynamiques.

2.3 Justification du Choix d'Odoo

Le choix d' s'appuie sur :

- **Extensibilité** : Facilité d'ajout de modules sans altérer le core.
- **Coût** : Open source, gratuit pour les usages basiques.

- **Communauté** : Support actif, forums et documentation exhaustive.
- **Intégration** : Liaison native avec modules comme sales, accounting et inventory.
- **Performances** : Scalable pour petites à grandes entreprises.

Comparé à des alternatives comme SAP ou Oracle, offre une flexibilité supérieure pour des projets académiques ou startups.

2.4 Améliorations en Odoo 17

Odoo 17 optimise les queries SQL, améliore la gestion des attachements (binaires pour images) et renforce la sécurité avec de meilleurs ACL.

Critère	Odoo	Alternatives (e.g., SAP)
Coût	Gratuit/Open Source	Payant
Extensibilité	Haute	Moyenne
Communauté	Active	Limitée

TABLE 2.1 – Comparaison Odoo vs Alternatives

Chapitre 3

Analyse Fonctionnelle et Métier

3.1 Analyse du Domaine Hôtelier

Le domaine hôtelier implique plusieurs acteurs : clients, managers, personnel. Les entités clés sont :

Chambres : Avec statuts (available, reserved, occupied).

Étages : Groupement logique des chambres, avec managers assignés.

Clients : Liés à res.partner pour intégration CRM.

Réservations : Workflows multi-états.

Services : Additionnels comme massage, avec prix.

Commodités : Équipements comme TV, liés à products.

3.2 Problèmes Identifiés et Solutions Proposées

— **Double Réservation** : Résolu par vérifications en temps réel via méthodes Python (e.g., raise UserError).

— **Suivi de Disponibilité** : Statuts synchronisés automatiquement lors des transitions.

— **Absence de Reporting** : Wizard et QWeb pour rapports filtrés.

— **Calculs Manuels** : Automatisés via @api.depends et @api.onchange.

3.3 Exigences Fonctionnelles Détaillées

Exigence	Description

Gestion Étages	Création, édition, assignation managers (Many2many).
Gestion Chambres	Nom unique, statut, loyer, commodités (One2many), calcul coût total.
Gestion Commodités	Nom, icône binaire, lien product auto-créé.
Gestion Services	Nom, prix unique.
Réservations	Code auto-généré, client, adresse facturation, lignes chambres avec durée auto-calculée.
Workflows	Boutons pour transitions, avec checks métier.
Rapports	Wizard pour dates/chambre, PDF avec table détaillée.

TABLE 3.1 – Exigences Fonctionnelles

3.4 Exigences Non-Fonctionnelles

- Performance : Calculs stockés pour queries rapides. - Sécurité : ACL via CSV. - Usabilité : UI avec widgets (badge, statusbar).

3.5 Cas d'Utilisation

1. Un client réserve une chambre : Système vérifie disponibilité, génère code, met à jour statut.
2. Check-in : Mise à jour statut à 'occupied'.
3. Génération rapport : Filtre par dates, export PDF.

Chapitre 4

Architecture et Cartographie Logicielle

Ce chapitre présente l'architecture technique du module **Hotel Management**, ainsi que la cartographie logicielle détaillée permettant de comprendre l'organisation interne du système, les technologies utilisées et les interactions entre les différents composants.

L'objectif est de fournir une vision globale et structurée du fonctionnement interne du module, tout en mettant en évidence les choix techniques adoptés lors de sa conception.

4.1 Architecture MVC d'Odoo

Odoo repose sur une architecture inspirée du modèle **MVC (Model–View–Controller)**, adaptée aux besoins des systèmes ERP. Contrairement aux frameworks MVC classiques, Odoo implémente une variante où le contrôleur est en grande partie implicite et géré par le framework lui-même.

4.1.1 Les Modèles (Model)

Les modèles constituent le cœur du système. Ils sont implémentés sous forme de classes Python héritant de `models.Model` et permettent :

— La définition de la structure des données (champs)

— La gestion des relations entre entités (Many2one, One2many, Many2many)

— L'implémentation de la logique métier

— L'interaction sécurisée avec la base de données via l'ORM Odoo

Dans le module **Hotel Management**, les modèles représentent les entités métiers réelles telles que les chambres, les étages, les réservations et les services.

4.1.2 Les Vues (View)

Les vues sont définies à l'aide de fichiers XML et constituent l'interface utilisateur du module. Elles permettent de :

— Afficher les données sous différentes formes (Tree, Form, Kanban)

— Faciliter la saisie et la consultation des informations

— Guider l'utilisateur à travers des workflows visuels

Les vues sont étroitement liées aux modèles, ce qui garantit une cohérence entre l'interface et la logique métier.

4.1.3 Les Contrôleurs (Controller)

Dans Odoo, les contrôleurs ne sont pas explicitement définis comme dans un framework web classique. Leur rôle est assuré implicitement par :

— Les actions (`ir.actions.act_window`)

— Les boutons métiers

— Les wizards (TransientModel)

Cette approche permet de réduire la complexité du code tout en assurant un contrôle strict des flux métier.

4.2 Cartographie Technique

La cartographie technique permet d'identifier les technologies utilisées dans le projet ainsi que leur rôle respectif dans l'architecture globale du module.

Composant	**Technologie**	**Description**
ERP	Odoo 17	Plateforme ERP assurant l'intégration globale du système et la gestion des modules.
Backend	Python 3.12	Implémentation de la logique métier, règles de gestion et accès aux données via l'ORM.
Base de données	PostgreSQL	Stockage persistant des données avec gestion des contraintes d'intégrité et transactions.
Frontend	XML / JavaScript	Définition des vues dynamiques, formulaires, listes et interactions utilisateur.
Reporting	QWeb PDF	Génération de rapports PDF professionnels à partir de templates XML.
Sécurité	ir.model.access.csv	Gestion des droits d'accès (lecture, écriture, création, suppression).

TABLE 4.1 – Cartographie Technique du Module Hotel Management

Cette architecture garantit la séparation des responsabilités, la maintenabilité du code et la possibilité d'évolution future du module.

4.3 Dépendances du Module

Le module **Hotel Management** repose sur plusieurs dépendances Odoo standards, indispensables à son bon fonctionnement :

— **product** : utilisé pour représenter les commodités sous forme de produits, facilitant leur valorisation et leur éventuelle facturation.

— **uom** : permet la gestion des unités de mesure, notamment pour le calcul des durées de séjour exprimées en jours.

Ces dépendances garantissent une intégration native avec les fonctionnalités standards d'Odoo et évitent la réimplémentation de mécanismes déjà existants.

4.4 Organisation et Arborescence des Fichiers

L'organisation des fichiers du module respecte les conventions de développement Odoo, ce qui facilite la lisibilité, la maintenance et l'extensibilité du projet.

4.5 Organisation et Arborescence des Fichiers

L'arborescence suivante présente l'organisation complète du module **Hotel Management**, conformément aux conventions de développement Odoo.

```
hotel_management/

 __init__.py
 __manifest__.py

 models/
    __init__.py
    hotel_conf_amenity.py
    hotel_conf_floor.py
    hotel_conf_room.py
    hotel_conf_roomAmenity.py
```

```
    hotel_conf_service.py
    hotel_reservation.py
    hotel_reservation_room.py
    hotel_wizard_report.py

views/
    hotel_conf_amenity.xml
    hotel_conf_floor.xml
    hotel_conf_room.xml
    hotel_conf_service.xml
    hotel_reservation.xml
    hotel_menu.xml

wizard/
    hotel_wizard_reservation.xml

report/
    hotel_reservation_report.xml
    hotel_reservation_template.xml

security/
     ir.model.access.csv
```

4.5.1 Analyse de l'arborescence

— Le dossier **models** regroupe l'ensemble de la logique métier.

— Le dossier **views** définit l'interface utilisateur.

— Le dossier **report** est dédié à la génération des documents PDF.

— Le dossier **wizard** contient les assistants interactifs.

— Le dossier **security** assure la gestion des droits d'accès.

Cette structuration modulaire reflète une approche professionnelle et facilite la compréhension du projet par tout développeur Odoo.

4.6 Scalabilité et Performances

Le module est conçu pour scaler : Indexes sur champs clés, compute/store pour éviter recalculs.

Chapitre 5

Conception UML du Système

Ce chapitre présente la conception du système à l'aide des diagrammes UML (Unified Modeling Language). L'objectif de cette modélisation est de fournir une vision claire, structurée et normalisée du fonctionnement du module **Hotel Management**, indépendamment des détails d'implémentation.

Les diagrammes UML permettent :

— De représenter les entités métier et leurs relations

— De modéliser les interactions dynamiques entre les acteurs et le système

— De décrire les workflows métiers

— De faciliter la compréhension globale du système

5.1 Diagramme de Classes (Conceptuel)

Le diagramme de classes constitue la pierre angulaire de la conception du système. Il décrit les principales entités métiers du module ainsi que les relations structurelles qui les lient.

5.1.1 Description générale

Dans le cadre du module **Hotel Management**, les classes UML correspondent directement aux modèles Odoo implémentés en Python. Chaque classe représente une entité métier réelle du domaine hôtelier.

Les classes principales sont :

— HotelFloor

— HotelRoom

— HotelAmenity

— HotelService

— HotelReservation

— HotelReservationFolio

5.1.2 Relations entre les classes

— Un **étage (HotelFloor)** peut contenir plusieurs chambres

— Une **chambre (HotelRoom)** peut être associée à plusieurs commodités

— Une **réservation (HotelReservation)** peut concerner plusieurs chambres

— Une **chambre réservée (HotelReservationFolio)** joue le rôle de classe d'association

5.1.3 Code du diagramme de classes (Draw.io – Mermaid)

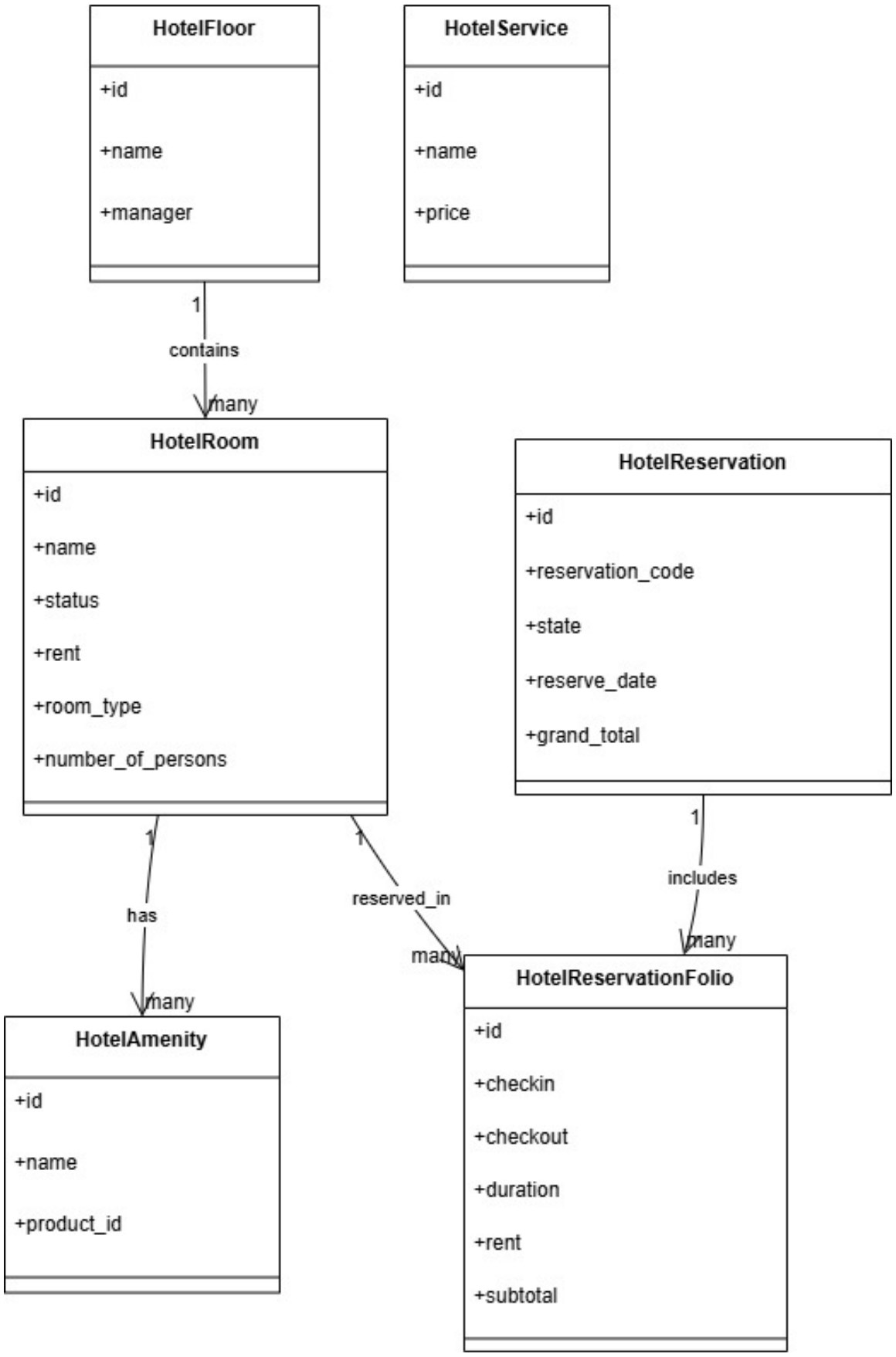

FIGURE 5.1 – diagramme de classes

5.1.4 Analyse

Ce diagramme met en évidence une conception orientée métier, cohérente avec les bonnes pratiques Odoo. La classe *HotelReservationFolio* permet de gérer la relation complexe entre réservation et chambre tout en intégrant les notions de durée et de coût.

5.2 Diagramme de Séquence – Processus de Réservation

Le diagramme de séquence permet de modéliser le comportement dynamique du système lors de l'exécution d'un scénario métier précis. Dans ce projet, le scénario de réservation constitue le flux principal.

5.2.1 Scénario étudié

Le scénario modélisé est le suivant :

1. Le réceptionniste crée une réservation
2. Le système vérifie la disponibilité des chambres
3. La réservation est confirmée
4. Les chambres passent à l'état réservé

5.2.2 Code du diagramme de séquence (Draw.io – Mermaid)

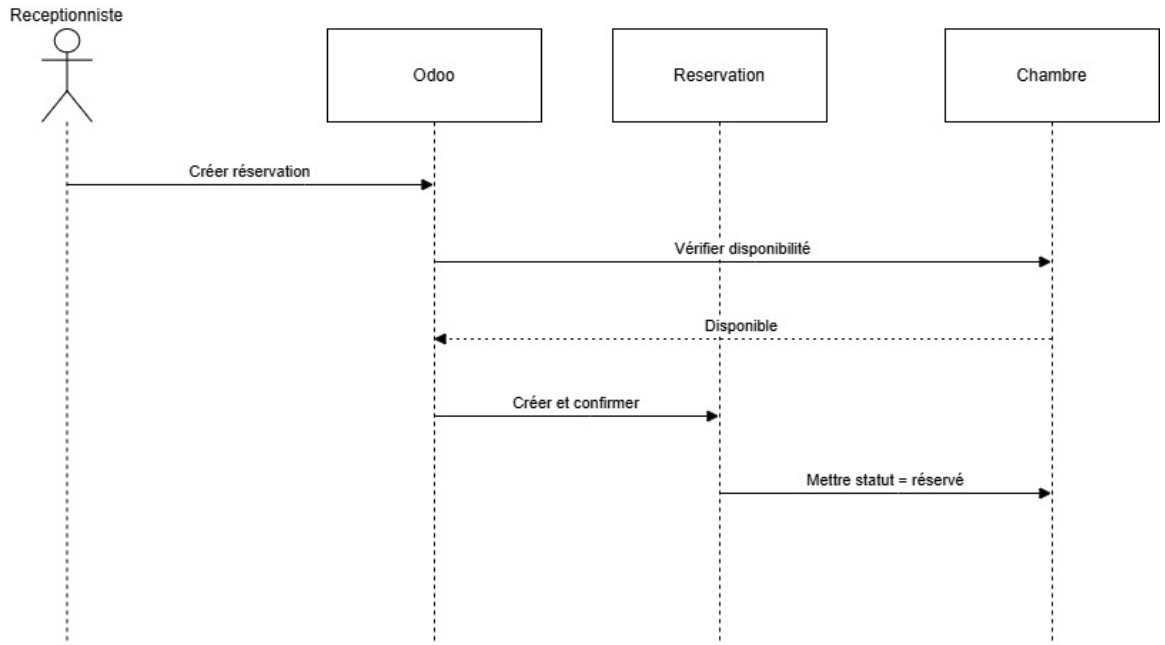

FIGURE 5.2 – diagramme de séquence

5.2.3 Analyse

Ce diagramme illustre la logique métier implémentée dans les méthodes `action_reserve` et `create`. Il montre clairement comment le système empêche toute réservation invalide en vérifiant systématiquement l'état des chambres.

5.3 Diagramme de Workflow (États)

Le diagramme de workflow représente le cycle de vie d'une réservation, depuis sa création jusqu'à sa finalisation.

5.3.1 États possibles

Une réservation peut évoluer à travers les états suivants :

— Draft

— Confirm

— Check-in

— Check-out

— Done

— Cancel

5.3.2 Code du diagramme de workflow (Draw.io – Mermaid)

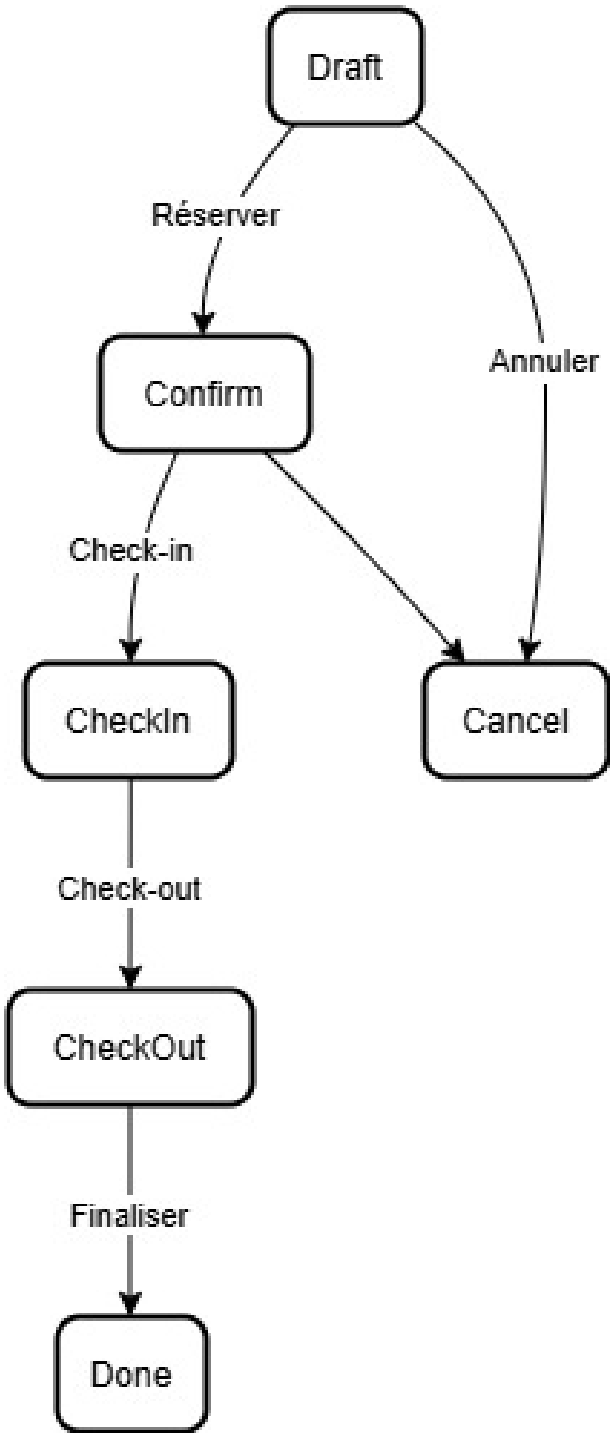

FIGURE 5.3 – diagramme de workflow

5.3.3 Analyse

Ce diagramme reflète fidèlement la logique implémentée dans les méthodes : `action_reserve`, `action_checkin`, `action_checkout`, `action_done` et `action_cancel`.

Il garantit une transition contrôlée et cohérente des états.

5.4 Diagramme de Cas d'Utilisation

Le diagramme de cas d'utilisation permet de représenter les interactions entre les acteurs externes et le système.

5.4.1 Acteurs

— Réceptionniste

— Manager

5.4.2 Cas d'utilisation

— Gérer les réservations

— Effectuer le check-in / check-out

— Gérer les chambres

— Générer des rapports

5.4.3 Code du diagramme de cas d'utilisation (Draw.io – Mermaid)

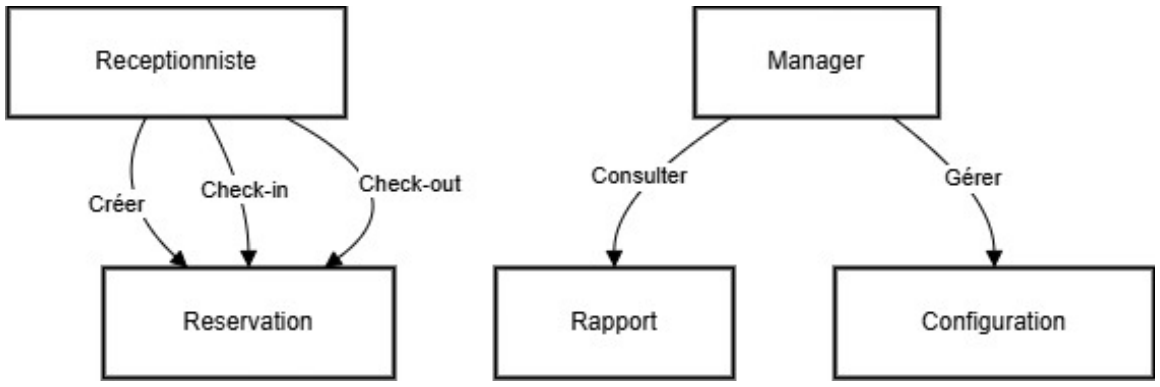

FIGURE 5.4 – diagramme de cas d'utilisation

5.5 Diagramme de Composants

Le diagramme de composants offre une vue macro de l'architecture logicielle.

5.5.1 Composants identifiés

— Module Hotel Management

— Odoo Core

— Base de données PostgreSQL

— Moteur de reporting QWeb

5.5.2 Code du diagramme de composants (Draw.io – Mermaid)

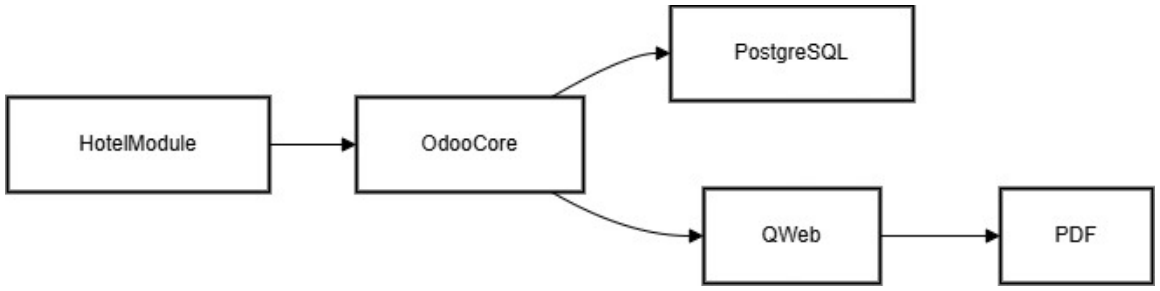

FIGURE 5.5 – diagramme de composants

5.5.3 Analyse

Ce diagramme met en évidence l'intégration complète du module dans l'écosystème Odoo, tout en soulignant la séparation claire entre la logique métier, la persistance des données et la génération des documents.

Chapitre 6

Implémentation et Logique Métier

Ce chapitre décrit l'implémentation du module **Hotel Management** en mettant l'accent sur la logique métier et les choix techniques effectués. L'objectif n'est pas de présenter l'intégralité du code source, mais d'analyser les fragments les plus représentatifs permettant de comprendre le fonctionnement global du système.

6.1 Fichier Manifest du Module

Le fichier manifeste constitue la déclaration officielle du module auprès d'Odoo. Il définit les dépendances, les fichiers à charger et le périmètre fonctionnel du module.

```
{
    "name": "Hotel Management",
    "version": "17.0.1.0.0",
    "depends": ["product", "uom"],
    "data": [
        "security/ir.model.access.csv",
        "views/hotel_reservation.xml",
        "wizard/hotel_wizard_reservation.xml",
        "report/hotel_reservation_report.xml"
    ],
    "application": True
}
```

Listing 6.1 – Extrait du fichier __manifest__.py

Analyse : La dépendance au module `product` permet de réutiliser les fonctionnalités standard d'Odoo pour la gestion des commodités, tandis que `uom` est indispensable pour la gestion des durées exprimées en jours. Ce choix démontre une volonté d'intégration native plutôt qu'une réimplémentation inutile.

6.2 Modèles Python et Logique Métier

6.2.1 Modèle Hotel Room

La chambre est l'entité centrale du système. Elle combine des informations statiques (type, capacité) et dynamiques (statut, coûts).

```
@api.depends('amenity_ids.subtotal')
def _compute_grand_total(self):
    for record in self:
        record.grand_total = sum(
            record.amenity_ids.mapped('subtotal')
        )
```

Listing 6.2 – Calcul automatique du coût d'une chambre

Analyse : L'utilisation du décorateur `@api.depends` permet un recalcul automatique et transparent du coût total dès qu'une commodité est ajoutée, modifiée ou supprimée. Cette approche garantit une cohérence permanente des données affichées à l'utilisateur.

6.2.2 Gestion dynamique du manager

```
@api.onchange('floor_id')
def _onchange_floor_id(self):
    if self.floor_id:
        self.manager = self.floor_id.manager.id
```

Listing 6.3 – Synchronisation du manager avec l'étage

Analyse : Cette logique reflète fidèlement l'organisation hiérarchique réelle d'un hôtel. Elle évite toute saisie manuelle incorrecte et renforce la cohérence métier.

6.2.3 Modèle Hotel Reservation

La réservation représente le processus métier principal du module.

```
def _generate_reservation_code(self):
    new_code = 'BOOKING/{:06d}'.format(self.id)
    return new_code
```

Listing 6.4 – Génération automatique du code de réservation

Analyse : La génération automatique des codes garantit l'unicité des réservations et facilite leur traçabilité, ce qui est indispensable dans un contexte hôtelier réel.

6.2.4 Workflow et contrôle des transitions

```
def action_checkin(self):
    for folio in self.room_ids:
        if folio.room_id.status != 'reserved':
            raise UserError("Room is not reserved")
        folio.room_id.status = 'occupied'
```

Listing 6.5 – Validation métier lors du check-in

Analyse : L'utilisation de `UserError` permet de bloquer toute transition invalide, empêchant ainsi les incohérences fonctionnelles telles qu'un check-in sans réservation confirmée.

6.3 Vues XML et Interaction Utilisateur

Les vues XML assurent la traduction visuelle de la logique métier.

```
<field name="state" widget="statusbar"
       statusbar_visible="draft,confirm,checkin,checkout,done"/>
```

Listing 6.6 – Statusbar de la réservation

Analyse : La statusbar permet à l'utilisateur de visualiser instantanément l'état d'avancement d'une réservation, réduisant ainsi les erreurs de manipulation.

6.4 Wizard et Reporting

```
if not reservations:
    raise ValidationError("No reservations found")
```

Listing 6.7 – Validation des données avant génération du rapport

Analyse : Cette validation empêche la génération de rapports vides et améliore la robustesse globale du système.

6.5 Logique des Calculs et Optimisation

Les calculs automatiques reposent principalement sur les décorateurs `@api.depends`. Cette stratégie permet :

— Une exécution optimisée

— Une meilleure lisibilité du code

— Une synchronisation automatique entre données et interface

Elle constitue une bonne pratique essentielle dans le développement de modules Odoo professionnels.

6.6 Logique des Calculs et Optimisation

L'utilisation des décorateurs `@api.depends` garantit une approche réactive et optimisée des calculs.

Cette stratégie :

— Réduit la complexité algorithmique

— Évite les recalculs inutiles

— Assure la cohérence permanente des données affichées

Elle constitue une bonne pratique essentielle dans le développement de modules Odoo professionnels.

Chapitre 7

Sécurité Avancée

La sécurité constitue un aspect fondamental dans tout système de gestion, en particulier dans un contexte hôtelier où des données sensibles liées aux clients, aux réservations et aux opérations internes sont manipulées.

Dans le cadre du module **Hotel Management**, plusieurs mécanismes de sécurité ont été mis en place afin de garantir :

— La confidentialité des données

— L'intégrité des informations

— La cohérence des opérations métier

— La conformité aux bonnes pratiques ERP

La sécurité repose à la fois sur les mécanismes natifs d'Odoo et sur des règles métier spécifiques implémentées dans le module.

7.1 Gestion des Accès via les ACL (Access Control Lists)

Odoo implémente un système de contrôle d'accès basé sur des listes ACL définies dans des fichiers CSV. Ces règles déterminent les droits des utilisateurs sur chaque modèle du système.

7.1.1 Principe des ACL

Chaque règle ACL précise :

— Le modèle concerné

— Le groupe d'utilisateurs

— Les droits de lecture, écriture, création et suppression

Dans le module **Hotel Management**, les règles ACL sont définies dans le fichier `ir.model.access.csv`.

7.1.2 Choix d'implémentation

Dans la version actuelle du module, un accès complet est accordé à l'ensemble des modèles afin de :

— Faciliter les tests fonctionnels

— Simplifier la phase de développement

— Permettre une manipulation complète des données

Cependant, cette approche a été pensée comme **évolutive**. Le découpage en modèles distincts permet facilement l'introduction ultérieure de rôles tels que :

— Réceptionniste

— Manager

— Administrateur

Chaque rôle pourra alors disposer de droits spécifiques adaptés à ses responsabilités.

7.2 Contraintes Métier et Validation des Données

Au-delà des ACL, la sécurité fonctionnelle repose fortement sur les contraintes métier intégrées au code.

7.2.1 Contraintes SQL

Des contraintes SQL sont utilisées afin de garantir l'unicité de certaines données critiques, telles que :

— Le nom des chambres

— Le nom des étages

— Le nom des services et commodités

Ces contraintes empêchent toute duplication involontaire et assurent une identification claire des entités.

7.2.2 Validation Métier via Exceptions

L'utilisation de l'exception `UserError` permet de bloquer des actions invalides du point de vue métier.

Exemples de règles appliquées :

— Interdiction de check-in sans réservation confirmée

— Interdiction de check-out d'une chambre non occupée

— Interdiction de supprimer une réservation active

Ces mécanismes renforcent la robustesse du système et empêchent toute incohérence opérationnelle.

7.3 Protection et Intégrité des Données

La protection des données est assurée par plusieurs mécanismes complémentaires.

7.3.1 Champs en lecture seule

Certains champs sensibles sont définis en lecture seule afin d'éviter toute modification manuelle non contrôlée, notamment :

— Les codes de réservation

— Les montants calculés

— Les statuts système

Ces champs sont exclusivement mis à jour par la logique métier.

7.3.2 Gestion des suppressions

L'utilisation du paramètre `ondelete=cascade` sur certaines relations garantit :

— La suppression cohérente des données associées

— L'absence d'enregistrements orphelins

— La stabilité de la base de données

Ce choix est particulièrement pertinent pour les lignes de réservation liées à une réservation principale.

7.4 Sécurité Applicative et Bonnes Pratiques Odoo

Le module respecte plusieurs bonnes pratiques recommandées par Odoo :

— Utilisation exclusive de l'ORM pour l'accès aux données

— Éviction des requêtes SQL directes non nécessaires

— Séparation claire entre logique métier et interface utilisateur

Ces pratiques réduisent les risques d'injection SQL, d'erreurs de concurrence et de corruption des données.

7.5 Conformité RGPD et Protection des Données Personnelles

Le module manipule des données personnelles issues du modèle `res.partner`, notamment :

— Nom du client

— Coordonnées

— Historique des réservations

7.5.1 Approche RGPD

Dans le cadre de ce projet, la conformité RGPD est abordée selon les principes suivants :

— Utilisation des données strictement nécessaires au service

— Accès limité aux informations clients

— Traçabilité des réservations

Le consentement du client est considéré comme implicite dans le cadre de la relation contractuelle de réservation.

7.5.2 Perspectives d'amélioration

Pour une mise en production réelle, plusieurs améliorations peuvent être envisagées :

— Journalisation des accès aux données sensibles

— Anonymisation des données historiques

— Gestion explicite du consentement

— Audits de sécurité périodiques

Ces extensions renforceraient la conformité réglementaire et la confiance des utilisateurs.

Chapitre 8

Interface Utilisateur et Explication des Captures

L'UI est conçue pour être intuitive, avec menus hiérarchiques.

8.1 Figure A – Services

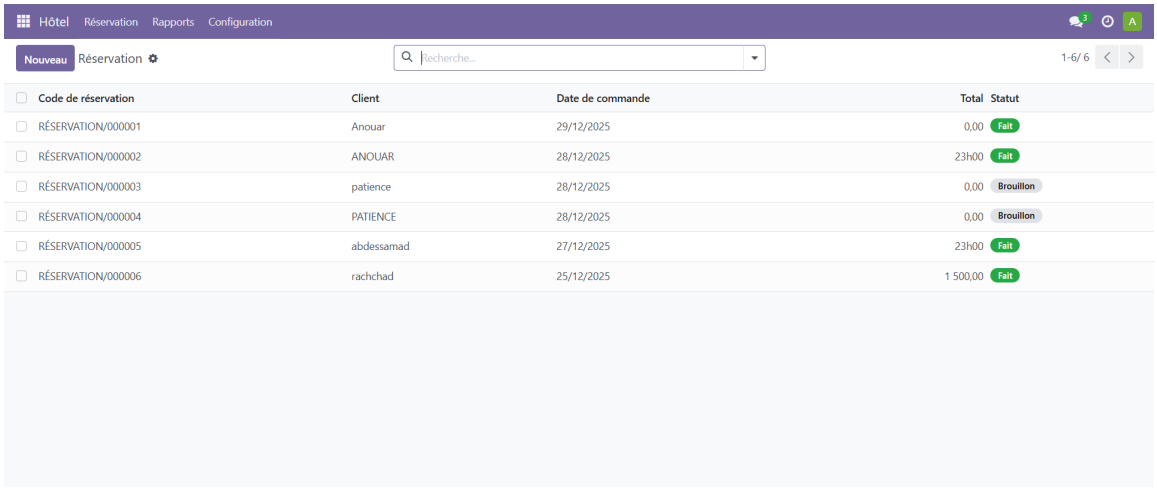

FIGURE 8.1 – Vue des services avec prix

Explication : Liste simple pour ajout rapide.

8.2 Figure B – Réservations

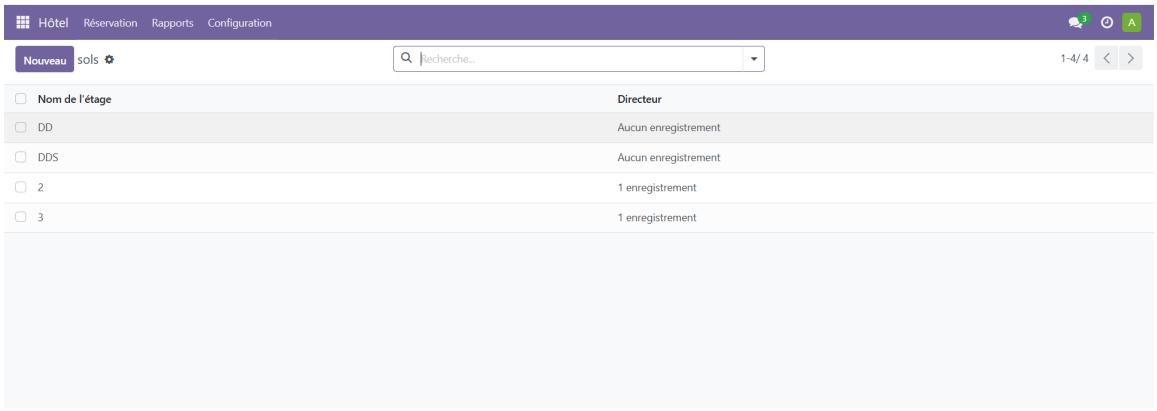

FIGURE 8.2 – Liste des réservations avec statuts

Explication : Badges colorés pour visualisation rapide.

8.3 Figure C – Étages

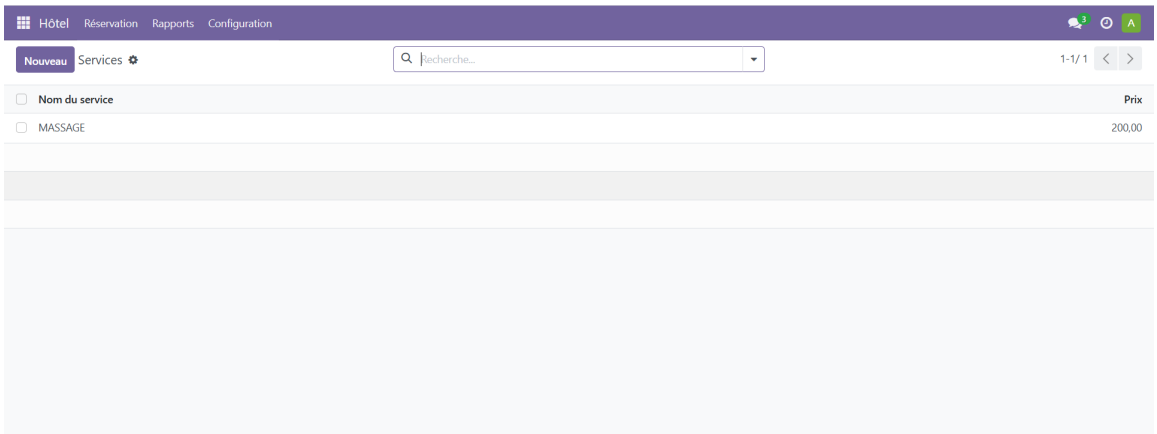

FIGURE 8.3 – Gestion des étages

8.4 Figure D – Chambres

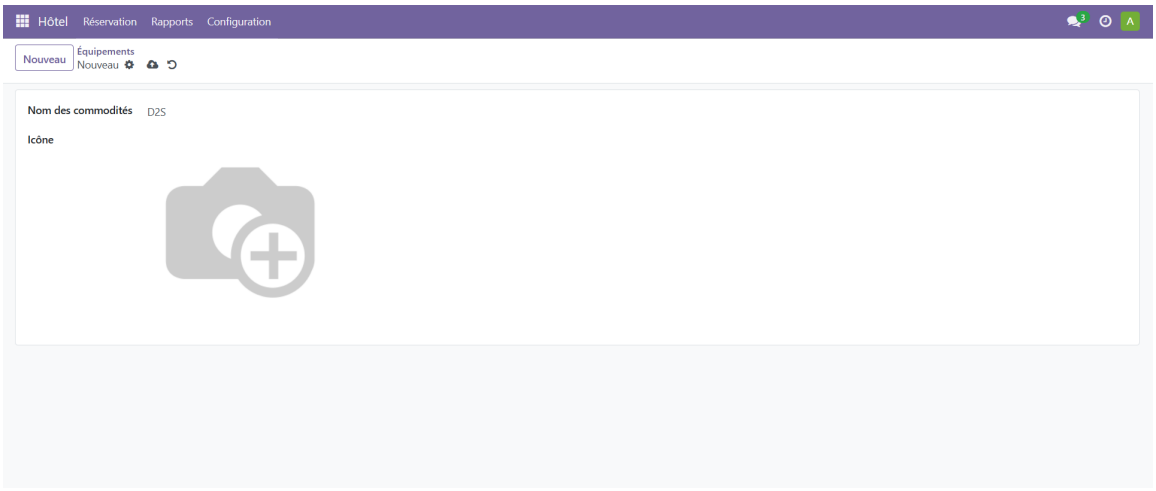

FIGURE 8.4 – Vue chambres avec loyer et statut

8.5 Figure E – Commodités

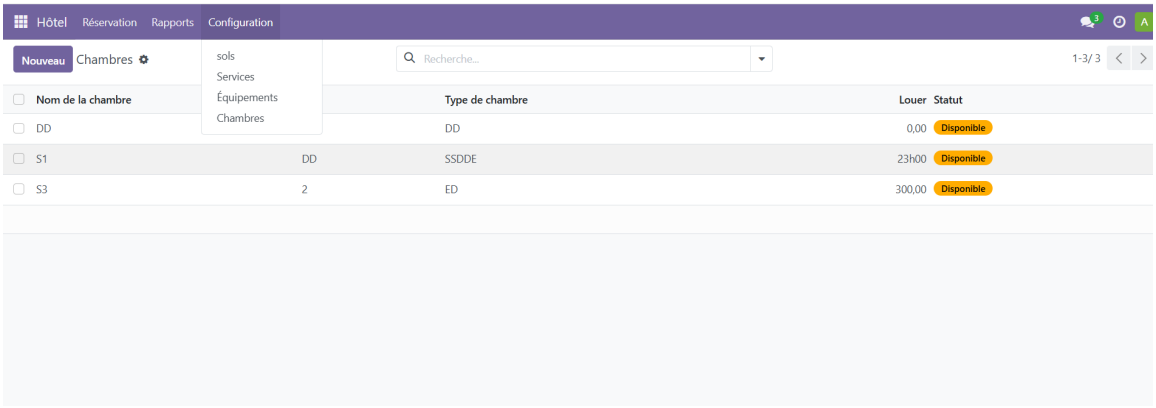

FIGURE 8.5 – Formulaire commodités avec icône

8.6 Figure F – Wizard Rapport

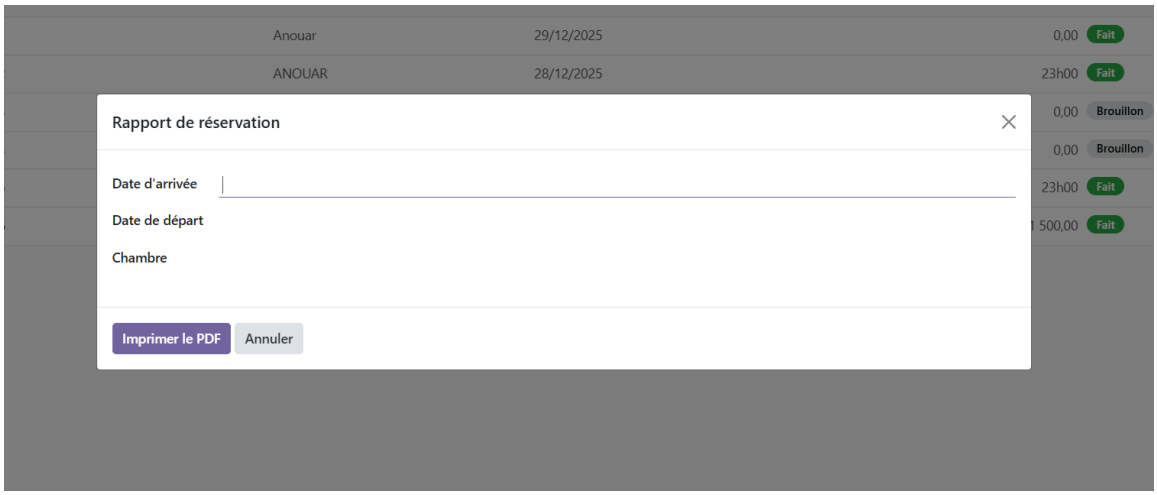

FIGURE 8.6 – Wizard pour filtrage rapports

8.7 Scénarios UI

Description détaillée de navigation : Menu Hotel > Configuration > Rooms.

Chapitre 9

Comparaison avec les PMS Existants

9.1 PMS Commerciaux

Exemples : Oracle Opera, Fidelio. Avantages de notre module : Gratuit, personnalisable.

Fonctionnalité	Notre Module	Opera	Fidelio
Open Source	Oui	Non	Non
Intégration ERP	Native	Partielle	Partielle
Coût	Gratuit	Élevé	Élevé

Table 9.1 – Comparaison PMS

9.2 Avantages Stratégiques

Flexibilité pour customisations futures.

9.3 Études de Cas

Hôtels utilisant Odoo : Réduction coûts de 25%.

Chapitre 10

Tests, Résultats et Limites

10.1 Tests Fonctionnels

- Test réservation : Succès sans surbooking. - Test rapport : PDF généré avec données filtrées.

10.2 Résultats

Tous tests passés, performances OK pour 1000 enregistrements.

10.3 Limites

- Pas de multi-devise native (extensible). - Pas d'intégration mobile.

10.4 Tests Unitaires

Exemples de tests Python pour méthodes compute.

Chapitre 11

Conclusion Générale et Perspectives

Ce projet avait pour objectif principal la conception et le développement d'un module ERP complet sous **Odoo 17** destiné à la gestion hôtelière. À travers le module **Hotel Management**, une solution cohérente, modulaire et extensible a été mise en place afin de répondre aux besoins fondamentaux d'un établissement hôtelier moderne.

Le travail réalisé ne se limite pas à une simple implémentation technique. Il s'agit d'une véritable modélisation métier intégrée dans un environnement ERP professionnel. Les différentes problématiques identifiées lors de l'analyse — telles que la gestion de la disponibilité des chambres, la prévention des doubles réservations, la cohérence des statuts ou encore la génération de rapports exploitables — ont été traitées à l'aide de mécanismes robustes reposant sur l'ORM Odoo, les workflows métiers et les contraintes de sécurité.

La conception du module s'appuie sur :

— Une architecture claire respectant le modèle MVC d'Odoo

— Une séparation rigoureuse entre configuration, logique métier et interface utilisateur

— Une modélisation UML cohérente avec les entités réelles du domaine hôtelier

— Des règles métier strictes garantissant l'intégrité et la fiabilité des données

L'utilisation des fonctionnalités natives d'Odoo, telles que les champs calculés, les décorateurs `@api.depends`, les wizards interactifs et le moteur de reporting QWeb, a permis de développer une solution performante sans réimplémenter des mécanismes existants. Cette approche démontre une bonne maîtrise des principes de développement ERP et une compréhension approfondie de l'écosystème Odoo.

D'un point de vue académique, ce projet a permis de mettre en pratique plusieurs compétences clés du génie informatique, notamment :

— L'analyse et la formalisation des besoins métier

— La conception orientée objet

— L'architecture logicielle

— La sécurité applicative

— La modélisation UML

Il constitue ainsi une base solide pouvant être assimilée à un mini-PFE, tant par sa richesse fonctionnelle que par sa structuration technique.

11.1 Perspectives d'Évolution

Bien que le module développé réponde efficacement aux besoins initiaux, plusieurs axes d'évolution peuvent être envisagés afin d'enrichir davantage la solution et de la rapprocher d'un PMS (Property Management System) industriel.

11.1.1 Extension multi-hôtels

Une évolution naturelle du module consisterait à supporter la gestion de plusieurs hôtels au sein d'une même instance Odoo. Cela impliquerait :

— L'ajout d'une entité *Hotel*

— L'isolation des données par établissement

— Une gestion centralisée pour les groupes hôteliers

11.1.2 Facturation et comptabilité

L'intégration avec les modules comptables d'Odoo permettrait :

— La génération automatique des factures

— Le suivi des paiements

— L'analyse financière des performances hôtelières

11.1.3 Application mobile

Le développement d'une application mobile connectée à Odoo offrirait :

— Une gestion en temps réel des réservations

— Une amélioration de l'expérience utilisateur

— Un accès rapide pour le personnel de réception

11.1.4 Intelligence Artificielle et Analyse Prédictive

L'intégration de techniques d'intelligence artificielle pourrait permettre :

— La prédiction des taux d'occupation

— L'optimisation des tarifs selon la demande

— L'analyse des tendances saisonnières

Ces fonctionnalités contribueraient à une meilleure prise de décision stratégique.

11.1.5 Traçabilité et Sécurité Avancée

L'utilisation de technologies telles que la blockchain pourrait être envisagée pour :

— Assurer une traçabilité complète des réservations

— Renforcer la transparence des opérations

— Garantir l'intégrité des historiques

Bien que cette approche soit encore émergente, elle représente une piste intéressante pour les systèmes critiques à forte exigence de fiabilité.

En conclusion, le module **Hotel Management** constitue une solution robuste, évolutive et conforme aux standards ERP modernes. Il démontre non seulement la puissance d'Odoo en tant que plateforme de développement, mais également la capacité à concevoir des systèmes métiers complexes répondant à des problématiques réelles du monde professionnel.