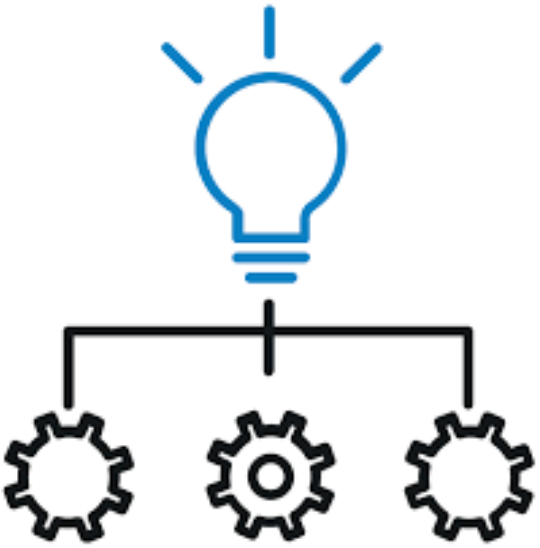
epsi® l'école d'ingénierie informatique

Projet Architecture Applicative

Documentation



Mehdi Sahari
Zakaria Talbi

EPSI 2024

Sommaire :

Introduction

Ce document présente une analyse détaillée de l'application de gestion d'hôtel, en se basant sur le code fourni pour les repositories, les modèles et les controllers. L'objectif est de décrire l'architecture de l'application, les patterns utilisés, les défis rencontrés et les solutions adoptées.

I. Architecture

L'application suit une architecture en couches, composée de trois couches principales :

Couche Repository: Cette couche fournit des abstractions pour accéder aux données persistantes, telles que les chambres, les clients et les réservations. Elle utilise le pattern Repository pour encapsuler l'accès aux données et promouvoir la séparation des préoccupations.

Couche Modèle: Cette couche représente les entités métier de l'application, telles que les chambres, les clients et les réservations. Les modèles encapsulent les données et le comportement des entités, définissant leurs propriétés et leurs méthodes.

Couche Controller: Cette couche gère les interactions avec l'utilisateur et orchestre les opérations métier. Les controllers utilisent les services de la couche métier pour effectuer des tâches telles que la réservation de chambres, l'annulation de réservations et la gestion des clients.

II. Patterns d'architecture

L'application utilise les patterns d'architecture suivants :

Pattern Repository: Ce pattern permet de séparer l'accès aux données du reste de l'application. Il définit une interface pour interagir avec les données, permettant aux autres couches de l'application de travailler avec des objets abstraits plutôt que de s'encombrer des détails d'implémentation spécifiques.

Pattern MVC (Model-View-Controller): Ce pattern structure l'application en trois couches distinctes : le modèle, la vue et le contrôleur. Chaque couche a sa propre responsabilité, ce qui favorise la modularité et la maintenabilité du code. Ici on ne possède pas de vues car tout se fait uniquement dans le terminale, nous n’avons pas d’interface.

III. Défis et solutions

1. Gestion des Annulations de Réservation
Défi : L'un des défis majeurs était de gérer les annulations de réservation avec remboursement potentiel. Cela nécessitait une logique complexe pour déterminer si un remboursement devait être effectué en fonction de certains critères.

Solution : Dans le contrôleur ReceptionistController, une méthode a été implémentée pour gérer l'annulation d'une réservation avec la possibilité de remboursement.

2. Gestion de l'État des Chambres
Défi : Un autre défi était de maintenir un suivi précis de l'état des chambres, notamment pour la gestion du nettoyage et de la disponibilité.

Solution : Le modèle ChambreModel a été étendu pour inclure une énumération représentant l'état de la chambre. Cela a permis une gestion plus efficace de l'état des chambres et une meilleure prise de décision dans les opérations métier.

3. Gestion des Opérations de Nettoyage
Défi : Assurer une gestion efficace des opérations de nettoyage des chambres, y compris la priorisation des chambres à nettoyer en fonction de leur état et de leur disponibilité.

Solution : Le contrôleur MenageController a été mis en place pour permettre aux membres du personnel de ménage d'accéder aux chambres à nettoyer et de les marquer comme nettoyées une fois le travail terminé.

4. Gestion des Réservations Multiples
Défi : Permettre aux clients de réserver plusieurs chambres pour une même période, tout en évitant les conflits de réservation et en assurant la cohérence des données.

Solution : Le contrôleur ClientsController a été développé pour gérer la réservation de chambres par les clients, en vérifiant la disponibilité des chambres et en assurant la cohérence des réservations multiples.

5. Gestion des Arrivées et Départs
Défi : Assurer une gestion fluide des arrivées et des départs des clients, y compris la vérification de l'état des chambres, la gestion des paiements et la notification des dommages éventuels.

Solution : Le contrôleur ReceptionistsController offre des fonctionnalités pour gérer les arrivées et les départs des clients, en notant l'occupation des chambres, en gérant les paiements non effectués et en envoyant des notifications en cas de dommages.

IV. Documentation des modèles

ChambreModel: Représente une chambre d'hôtel avec ses attributs tels que le numéro, la disponibilité, le tarif, la capacité, le type et l'état.

TypeChambre: Énumération définissant les différents types de chambres (Simple, Double, Suite).

EtatChambre: Énumération définissant les différents états d'une chambre (Neuf, Refaite, ARefaire, RienASignaler, GrosDegats).

ClientModel: Représente un client de l'hôtel avec ses attributs tels que l'ID, le nom, le prénom, l'email et le numéro de téléphone.
MenageModel: Représente un membre du personnel de ménage avec ses attributs tels que l'ID et le nom.

ReservationModel: Représente une réservation faite par un client pour une chambre avec ses attributs tels que l'ID, le client, la chambre, la date de début, la date de fin et l'état d'annulation.

V. Documentation des controllers

L'application utilise plusieurs controllers pour gérer les différentes fonctionnalités de l'hôtel. Chaque controller est responsable d'un ensemble spécifique d'actions, regroupées par domaine métier.

ClientsController:

GetChambresDisponibles(DateTime dateDebut, DateTime dateFin): Permet de récupérer la liste des chambres disponibles pour les dates spécifiées.

ReserverChambre(int idClient, int idChambre, DateTime dateDebut, DateTime dateFin, string numeroCarte): Permet de réserver une chambre pour un client donné avec les détails de la réservation.

AnnulerReservation(int idReservation): Permet d'annuler une réservation existante.

MenageController:

GetChambresANettoyer(): Permet de récupérer la liste des chambres qui ont besoin d'être nettoyées.

MarquerNettoyage(int idChambre): Permet de marquer une chambre comme nettoyée après le ménage.

ReceptionistController:

GetChambresDisponibles(DateTime dateDebut, DateTime dateFin): Permet de récupérer la liste des chambres disponibles pour les dates spécifiées (même fonction que ClientsController).

AnnulerReservation(int idReservation, bool remboursement): Permet d'annuler une réservation existante, en gérant le remboursement potentiel.

GestionArrivee(int idReservation): Gère les actions à effectuer lors de l'arrivée d'un client (enregistrement, remise des clés, etc.).

GestionDepart(int idReservation): Gère les actions à effectuer lors du départ d'un client (paiement, état de la chambre, etc.).

ReservationController:

GetChambresANettoyer(): Permet de récupérer la liste des chambres qui ont besoin d'être nettoyées.

MarquerNettoyage(int idChambre): Permet de marquer une chambre comme nettoyée après le ménage.

Conclusion

Dans le cadre de ce projet, nous avons réalisé une architecture solide pour notre application de gestion d'hôtel, en utilisant une approche en couches avec des modèles, des repositories et des contrôleurs bien structurés. Les patterns d'architecture tels que Repository et MVC ont été mis en œuvre pour favoriser la modularité, la réutilisabilité et la maintenabilité du code.

En résumé, bien que nous ayons accompli une partie importante du travail en établissant une architecture solide pour notre application, il reste encore des étapes à franchir pour la rendre pleinement fonctionnelle. Avec un travail supplémentaire sur l'API afin de la rendre fonctionnelle.