**Documentation Technique – Projet GestionHotel**

Présentation du projet :

GestionHotel est une application de gestion hôtelière développée en C# avec ASP.NET Core 8. Elle permet de gérer les chambres, les réservations, les clients ainsi que les signalements d’incidents. Le projet inclut également un système de notification par email pour améliorer l’expérience client.

Architecture logicielle :

Le projet adopte une **architecture en couches (layered architecture)** respectant les principes de **séparation des responsabilités (Separation of Concerns)** et de **Clean Architecture**.

Organisation des couches :

* GestionHotel.Domain : Définit les entités métiers (ex : Client, Chambre, Réservation, Signalement) et interfaces.
* GestionHotel.Application : Contient la logique métier (services, gestion des règles).
* GestionHotel.Infrastructure : Accès aux données (EF Core + DbContext), intégrations techniques.
* GestionHotel.Externals : Services externes comme les mock de paiements.
* GestionHotel.Apis : Couche de présentation (controllers REST, configuration Swagger, dépendances DI, etc.).

Avantages du découplage

* Testabilité unitaire facilitée
* Scalabilité du code
* Meilleure maintenabilité
* Remplaçabilité des implémentations (ex : EmailService)

Technologies utilisées :

|  |  |
| --- | --- |
| Technologie | Usage |
| ASP.NET Core 8 | API REST |
| Entity Framework | ORM / gestion de la base de données |
| SQL Server | SGBD relationnel |
| Swagger | Documentation interactive de l’API |
| Mailtrap | Envoi de mails en environnement de test |
| C# | Langage principal |
| Serilog / Logging | Journalisation des erreurs et actions |

Fonctionnalités principales :

Gestion des chambres

* Ajouter / Modifier / Supprimer une chambre
* Consulter les chambres disponibles

Gestion des clients

* Ajouter / Modifier / Supprimer un client
* Associer un client à une réservation

Gestion des réservations

* Créer une réservation
* Check-in / Check-out
* Annulation par le client ou la réception
* Liste des réservations actives

Signalements d’incidents

* Déclarer un problème dans une chambre
* Marquer un signalement comme traité
* Supprimer un signalement

Notifications automatiques

* Avant séjour : rappel 24h avant la date d’arrivée
* Après séjour : remerciement + demande de feedback

Défis rencontrés & solutions apportées :

1. Concurrence sur les réservations

Problème : Possibilité de double-réservation de chambres.  
Solution : Ajout d’une logique métier côté Application.Services pour vérifier les disponibilités avant validation. Utilisation de la méthode ChambreService.GetChambresDisponibles.

2. Relation complexe Client ↔ Réservation ↔ Chambre

Problème : Propriété Client potentiellement null lors de la sérialisation.  
Solution : Utilisation d’Eager loading via .Include() dans les repositories pour éviter les erreurs de navigation EF Core. Ajout de vérifications null-safe dans les services.

3. Notification automatique (tâche planifiée)

Problème : Mise en œuvre de tâches de fond sans bloquer l’exécution de l’API.  
Solution : Implémentation de BackgroundService pour NotificationPreSejourService et NotificationPostSejourService. Utilisation d’un ServiceScopeFactory pour accéder aux services injectés.

4. Email non envoyé en environnement de dev

Problème : SMTP réel bloqué ou quota dépassé.  
Solution : Utilisation de Mailtrap pour tester sans envoyer d’emails réels. Ajout de logs pour traquer les échecs d'envoi (\_logger.LogWarning(...)).

5. Swagger et cohérence des routes

Problème : Swagger ne générait pas les bons schémas sans conventions.  
Solution : Structuration uniforme des routes ([Route("api/[controller]")]), validation des DTO, annotations pour rendre la doc Swagger complète.

Endpoints disponibles (Swagger) :

|  |  |  |
| --- | --- | --- |
| Méthode | Endpoint | Description |
| GET | /api/Chambres | Lister toutes les chambres |
| GET | /api/Chambres/disponibles | Chambres disponibles |
| POST | /api/Chambres | Ajouter une chambre |
| PUT | /api/Chambres/{id} | Modifier une chambre |
| DELETE | /api/Chambres/{id} | Supprimer une chambre |
| GET | /api/Clients | Lister les clients |
| POST | /api/Clients | Créer un client |
| PUT | /api/Clients/{id} | Modifier un client |
| DELETE | /api/Clients/{id} | Supprimer un client |
| POST | /api/Reservations | Créer une réservation |
| GET | /api/Reservations/actives | Voir les réservations actives |
| POST | /api/Reservations/checkin | Faire le check-in |
| POST | /api/Reservations/checkout | Faire le check-out |
| POST | /api/Reservations/annuler/{id} | Annuler par le client |
| POST | /api/Reservations/annuler-par-reception | Annuler par la réception |
| GET | /api/Signalements | Lister les signalements |
| POST | /api/Signalements | Créer un signalement |
| PUT | /api/Signalements/{id} | Marquer comme traité |
| DELETE | /api/Signalements/{id} | Supprimer un signalement |

Bonnes pratiques appliquées :

✅ Respect de l’architecture en couches  
✅ SOLID (principes de responsabilité unique, injection de dépendances, etc.)  
✅ Validation des entrées via les DTO  
✅ Logs structurés avec ILogger  
✅ Code asynchrone (async/await) pour toutes les opérations longues  
✅ Swagger pour tester et documenter l’API  
✅ Tests unitaires possibles grâce au découplage des interfaces