

# Gestion d'un système d'hôtellerie

Projet Ingénierie de données



**Réalise par :**

- MANSSOUR Youssef
- EL MOUDEN Hind

**Encadré par :**

- Prof CHERDAL Safae
- Prof DAALI Noussaiba

# Résumé

Dans le cadre de la **transformation numérique du secteur hôtelier**, ce projet propose la conception d'un **système de gestion de données distribué et évolutif**, capable de traiter efficacement des **flux de données temporelles à haute volumétrie**, tels que les historiques de séjours, les paiements, les factures et les indicateurs de performance hôtelière.

Afin de répondre aux exigences de **performance**, de **scalabilité horizontale** et de **tolérance aux pannes**, nous avons choisi d'utiliser **Apache Cassandra**, une **base de données NoSQL distribuée**, conçue pour supporter de lourdes charges d'écriture et assurer un accès rapide et fiable aux données, même en cas de forte sollicitation. Son architecture **peer-to-peer**, combinée à un mécanisme de **réplication automatique**, garantit une **haute disponibilité** des données.

Le système repose sur une **architecture full-stack moderne** :

- **Frontend** développé avec **React.js**, utilisant **Axios** pour la communication avec le serveur, et stylisé avec **Bootstrap** ou **TailwindCSS** pour assurer une interface utilisateur responsive et intuitive.
- **Backend** construit avec **Node.js** et **Express.js**, exposant une API RESTful qui interagit avec la base Cassandra via **CQL (Cassandra Query Language)**.
- **Docker** est utilisé pour conteneuriser et déployer l'environnement, notamment la base Cassandra, assurant ainsi portabilité, simplicité de configuration et déploiement multi-environnements.

Le système atteint plusieurs objectifs stratégiques :

- **Sauvegarde de l'historique détaillé des séjours clients** (hôtel, période, chambre, services utilisés).
- **Gestion des paiements** classés par type de service, période et établissement.
- **Archivage des factures** associées aux réservations, garantissant une traçabilité complète.
- **Génération de rapports analytiques** sur les taux d'occupation, les revenus et les tendances clients.
- **Optimisation de l'accès aux données historiques** pour des **analyses marketing avancées** (fidélisation, segmentation, prédiction).

Ce système a été conçu pour être **modulaire, évolutif et intégrable** à des solutions plus avancées, telles que des modules d'intelligence artificielle, des outils de visualisation dynamique (BI) ou encore des services tiers connectés. Il constitue ainsi une **solution robuste, moderne et adaptée aux enjeux futurs de la gestion hôtelière**.

## Sommaire

<b>Introduction .....</b>	<b>6</b>
<b>I) Objectif du projet .....</b>	<b>7</b>
<b>II) Technologies et outils utilisés .....</b>	<b>7</b>
a) Frontend.....	7
b) Backend.....	7
c) Base de Données.....	7
d) Conteneurisation .....	7
e) Environnement de développement .....	7
<b>III) Architecture de l'application.....</b>	<b>8</b>
<b>IV) Fonctionnalités développées .....</b>	<b>8</b>
<b>VI) Structure du projet .....</b>	<b>9</b>
1) <i>Difficultés rencontrées et solutions .....</i>	<i>11</i>
<b>VII) Présentation de Cassandra .....</b>	<b>11</b>
1) <i>Pourquoi Cassandra ?.....</i>	<i>11</i>
2) <i>Origine de Cassandra.....</i>	<i>12</i>
3) <i>Remarque: Les concepts de Cassandra.....</i>	<i>12</i>
4) <i>Avantages / Inconvénients.....</i>	<i>12</i>
a) <i>Avantages : .....</i>	<i>12</i>
b) <i>Inconvénients : .....</i>	<i>12</i>
<b>VIII) Architecture de Cassandra .....</b>	<b>13</b>
1) <i>Définition: Cluster.....</i>	<i>13</i>
2) <i>Fondamental:.....</i>	<i>13</i>
3) <i>Colonne.....</i>	<i>13</i>
a) <i>Remarque: .....</i>	<i>13</i>
4) <i>Définition: Ligne .....</i>	<i>13</i>
a) <i>Remarque: La notion de clé dans Cassandra .....</i>	<i>14</i>
5) <i>Définition: Column Family.....</i>	<i>14</i>
a) <i>Remarque: .....</i>	<i>15</i>
6) <i>Définition: Keyspace .....</i>	<i>16</i>
<b>IX) Gestion des données temporelles et financières.....</b>	<b>17</b>
1) <i>Historique des séjours et transactions financières .....</i>	<i>17</i>
2) <i>Optimisation des performances et scalabilité.....</i>	<i>17</i>
3) <i>Stratégies de modélisation temporelle dans Cassandra.....</i>	<i>17</i>

4) Utilisation des clés composites.....	18
5) Scalabilité horizontale .....	18
<b>X) Fonctionnalités principales du système .....</b>	<b>18</b>
2) Sauvegarde de l'historique des séjours par hôtel et période.....	19
3) Gestion des paiements clients par période et type de service.....	19
4) Stockage et analyse des données historiques .....	20
<b>XI) Avantages et perspectives .....</b>	<b>21</b>
1) Avantages de l'approche choisie.....	21
a) Scalabilité horizontale .....	21
b) Haute disponibilité .....	21
c) Performance sur les écritures.....	21
d) Modèle flexible .....	21
e) Adaptation aux données temporelles .....	21
2) Possibilités d'extensions et analyses avancées .....	21
a) Ajout de modules fonctionnels.....	21
c) Analyse prédictive .....	22
d) Tableaux de bord dynamiques.....	22
e) Vision future du projet .....	22
<b>XII) Expérimentation et discussions des résultats.....</b>	<b>22</b>
1) La page d'accueil.....	22
2) Clients.....	23
3) Pagination.....	23
5) Gestion des factures.....	24
6) Rapports statistiques.....	24
7) L'ajout d'un nouveau paiement.....	25
8) Gestion des paiements .....	25
9) Gestion des Séjours .....	26
10) Table factures .....	26
11) Table paiements.....	27
12) Table clients.....	27
13) Table statuts période.....	28
14) Factures par client.....	29
15) Facture par séjours.....	30
<b>Conclusion.....</b>	<b>31</b>

## Table des illustrations

Figure 1: Architecture de l'application .....	8
Figure 2: Colonne .....	13
Figure 3: Exemple colonne .....	13
Figure 4: Ligne .....	13
Figure 5: Exemple de ligne .....	14
Figure 6: Column family .....	14
Figure 7: Exemple de column family .....	15
Figure 8: Keyspace .....	16
Figure 9: La page d'accueil .....	22
Figure 10: clients .....	23
Figure 11: pagination .....	23
Figure 12: gestion des factures .....	24
Figure 13: Rapports statistiques .....	24
Figure 14: l'ajout d'un nouveau paiement .....	25
Figure 15: Gestion des paiements .....	25
Figure 16: Gestion des Séjours .....	26
Figure 17: table factures .....	27
Figure 18: Table paiements .....	27
Figure 19: Table clients .....	28
Figure 20: statut périodes .....	28
Figure 21: Factures par client .....	29
Figure 22: Facture par séjours .....	30

## Introduction

L'industrie hôtelière moderne fait face à une transformation numérique majeure, nécessitant la gestion simultanée de flux de données complexes et hétérogènes : réservations multi-canaux, profils clients personnalisés, inventaire dynamique, optimisation tarifaire et retours d'expérience. Cette complexité opérationnelle, amplifiée par les exigences de performance temps réel et la saisonnalité du secteur, révèle les limites des architectures relationnelles traditionnelles face à l'hétérogénéité des données (structurées, semi-structurées et complexes).

Ce projet vise à analyser et démontrer comment les technologies NoSQL peuvent répondre aux défis spécifiques de l'industrie hôtelière, en proposant une architecture polyglotte exploitant les forces complémentaires des différents paradigmes NoSQL. L'objectif consiste à concevoir une solution intégrant bases documentaires (profils clients), systèmes clé-valeur (sessions et cache), bases graphes (recommandations personnalisées) et solutions colonnaires (analytique temps réel) pour optimiser performance, scalabilité et personnalisation de l'expérience client.

Cette approche permettra d'évaluer l'adéquation des technologies NoSQL aux cas d'usage hôteliers critiques, de proposer une architecture technique détaillée et de quantifier les bénéfices métier en termes d'amélioration de l'expérience client et d'efficacité opérationnelle.

## I) Objectif du projet

L'objectif principal de ce projet est de concevoir une solution logicielle efficace et conviviale qui permet la gestion des différentes entités d'un hôtel. L'application doit permettre aux utilisateurs (gérants ou personnel) de gérer les **clients**, **chambres**, **réservations**, **séjours**, **paiements**, et **factures**, tout en garantissant une interface fluide et un backend performant.

## II) Technologies et outils utilisés

La réalisation de cette application s'est appuyée sur plusieurs technologies réparties entre le **frontend**, le **backend**, la **base de données**, et les **outils de développement**.

### a) Frontend

Pour la partie visible par l'utilisateur, nous avons utilisé **React.js**, une bibliothèque JavaScript moderne qui permet de créer des interfaces utilisateurs interactives et dynamiques. Le code est écrit en **JavaScript ES6**, en utilisant des composants fonctionnels, et en gérant l'état grâce aux **hooks** comme `useState` et `useEffect`.

La structure des pages est assurée par **HTML5**, et la présentation visuelle est gérée via **CSS3**. Pour les appels API vers le backend, nous avons utilisé **Axios**, qui facilite les échanges de données en HTTP. Pour un design responsive, nous avons également intégré un framework CSS moderne, comme **Bootstrap**.

### b) Backend

Le serveur backend est développé avec **Node.js**, un environnement JavaScript côté serveur, et le framework **Express.js** qui permet de créer facilement des routes RESTful. Le backend expose des points d'accès API tels que `/api/sejours`, `/api/clients`, ou `/api/factures`.

Nous avons utilisé le middleware **CORS** pour autoriser les appels depuis le frontend (port 3000) vers le backend (port 5000). Les données JSON envoyées par le frontend sont traitées automatiquement grâce à `express.json()`.

### c) Base de Données

Pour la persistance des données, nous avons utilisé **Apache Cassandra**, une base de données NoSQL distribuée. Elle permet de stocker efficacement les entités du projet (clients, chambres, séjours, etc.) tout en assurant de bonnes performances.

L'interaction avec Cassandra s'effectue via **CQL (Cassandra Query Language)** dans un terminal `cqlsh`, permettant de créer des `keyspaces`, des tables, et d'effectuer des opérations de lecture et d'écriture. Toutes les entités sont regroupées dans un `keyspace` nommé `hotel_management`.

### d) Conteneurisation

Pour éviter les problèmes d'installation manuelle de Cassandra, nous avons utilisé **Docker**, une technologie de conteneurisation. Grâce à Docker, il est possible de lancer un conteneur Cassandra prêt à l'emploi en quelques secondes. Cela permet aussi de garantir un environnement de travail stable et reproductible.

### e) Environnement de développement

Le développement a été réalisé avec **Visual Studio Code**, un éditeur de code puissant et adapté aux technologies JavaScript. L'installation des bibliothèques s'est faite via **npm**, le gestionnaire de paquets de Node.js. Les serveurs (frontend et backend) ont été lancés via le terminal PowerShell ou CMD. Pour le suivi des modifications de code, **Git** et **GitHub** peuvent être utilisés (selon les préférences de l'équipe).

### III) Architecture de l'application

L'architecture de l'application suit une approche **client-serveur** :

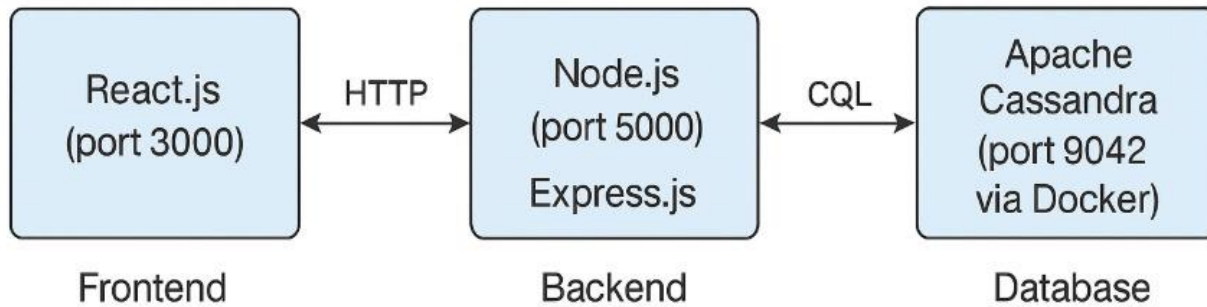


Figure 1: Architecture de l'application

- Le **frontend** envoie des requêtes vers l'API Express.
- Le **backend** traite les requêtes, effectue des opérations sur la base Cassandra, et renvoie les résultats au frontend.
- Les **données** sont stockées et récupérées dans des tables Cassandra via des requêtes CQL.

### IV) Fonctionnalités développées

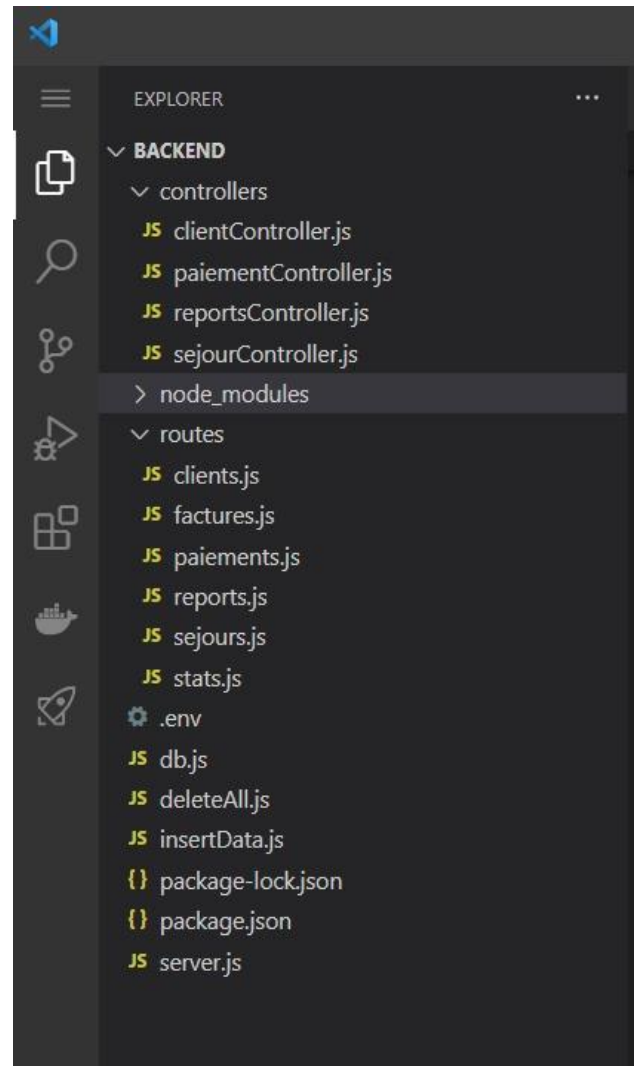
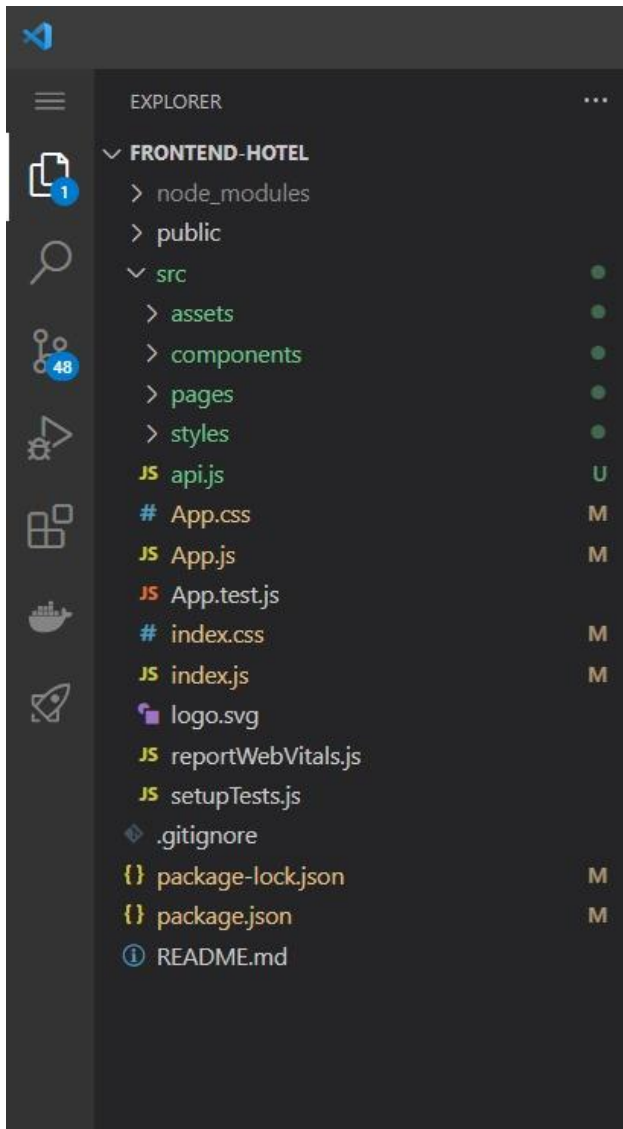
L'application comprend plusieurs modules fonctionnels, chacun correspondant à une entité métier de l'hôtel :

- **Gestion des clients** : ajout, modification, suppression, et consultation des fiches clients.
- **Gestion des chambres** : gestion des types, disponibilités et tarifs des chambres.
- **Réservations et séjours** : enregistrement des séjours avec dates d'arrivée et de départ, affectation à une chambre.
- **Paiements** : enregistrement des paiements liés à chaque séjour.
- **Factures** : génération et visualisation des factures clients.
- **Tableaux dynamiques** : affichage sous forme de tableaux avec tri, pagination, ou filtres selon les besoins.



## VI) Structure du projet

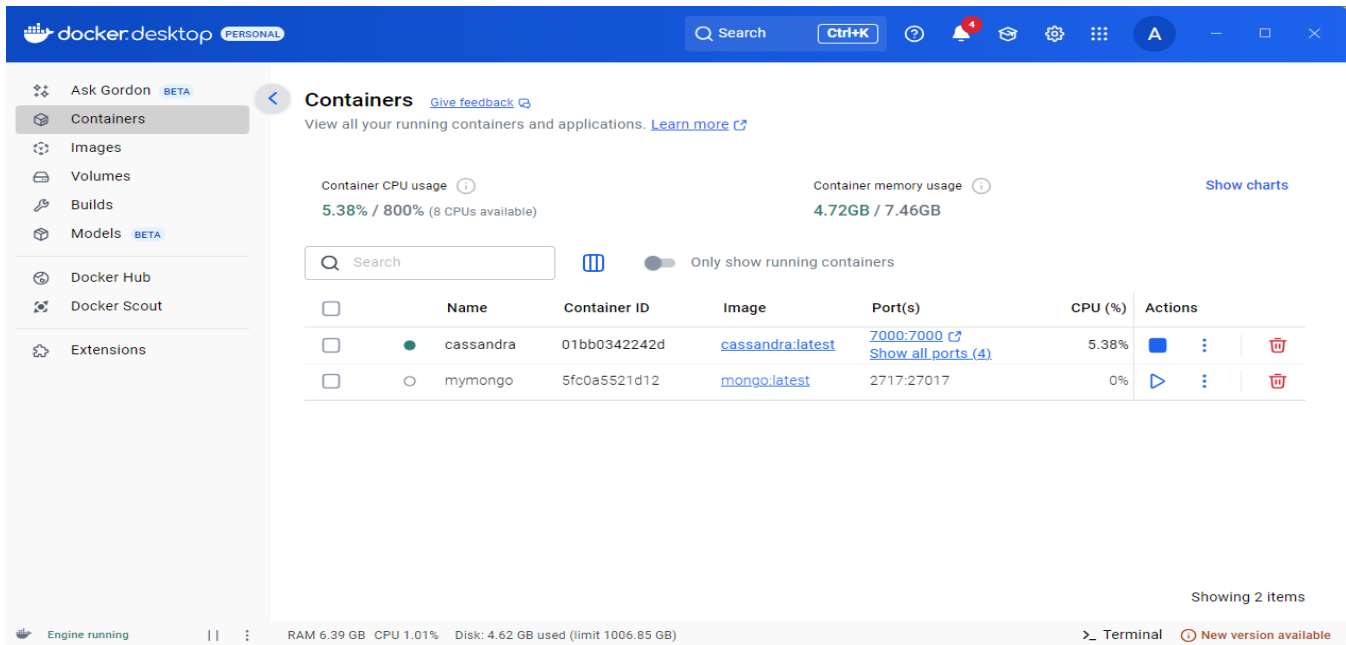
L'arborescence du projet se divise en deux grandes parties :



/docker

└─ cassandra

└─ docker-compose.yml



## Api.js entre backEnd et frondEnd

```
src > JS apijs > ...

1  import axios from 'axios';
2
3  const api = axios.create({
4    | baseURL: 'http://localhost:5000/api', // URL de ton backend
5  });
6  // Clients
7  export const getClients = (params) => api.get('/clients', { params });
8  export const searchClients = (term) => api.get('/clients/search?term=${term}');
9  export const deleteClient = (id) => api.delete('/clients/${id}');
10 export const getClientById = (id) => api.get('/clients/${id}');
11 export const addClient = (data) => api.post('/clients', data);
12 export const updateClient = (id, client) => api.put('/clients/${id}', client);
13 // Paiements
14 export const getPayments = () => api.get('/paiements');
15 export const addPayment = (data) => api.post('/paiements', data);
16 // Séjours
17 export const fetchSejours = () => api.get('/sejours');
18 export const addSejour = (sejour) => api.post('/sejours', sejour);
19 export const searchSejoursByClient = (clientId) => api.get('/sejours/search?client_id=${clientId}');
20 export const getFactures = (clientId) =>
21   | clientId ? api.get('/factures', { params: { clientId } }) : api.get('/factures');
22   | export const addFacture = (data) => api.post('/factures', data);
23   | export const updateFacture = (id, data) => api.put('/factures/${id}', data);
24   | export const deleteFacture = (id) => api.delete('/factures/${id}');
25 // Rapports
26 export const getPaymentsReport = (hotelId, start, end) =>
27   | api.get('/reports/paiements', { params: { hotelId, start, end } });
28 export const getClientsActivityReport = (hotelId, start, end) =>
29   | api.get('/reports/clients-activity', { params: { hotelId, start, end } });
30 export const getOccupationReport = (hotelId, startDate, endDate) =>
31   | api.get('/reports/occupation', { params: { hotel_id: hotelId, start_date: startDate, end_date: endDate } });
32 export const getTauxOccupation = (hotel_id, start_date, end_date) =>
33   | api.get('/reports/taux-occupation', { params: { hotel_id, start_date, end_date } });
34 export const getChiffreAffaires = (hotel_id, start_date, end_date) =>
35   | api.get('/reports/chiffre-affaires', { params: { hotel_id, start_date, end_date } });
36 export const getTotalSejours = (hotel_id, start_date, end_date) =>
37   | api.get('/reports/total-sejours', { params: { hotel_id, start_date, end_date } });
38 export const getPaiementsParType = (start_date, end_date) =>
39   | api.get('/reports/paiements-par-type', { params: { start_date, end_date } });
40 export const updatePayment = async (id, data) => axios.put('/paiements/${id}', data);
41 export default api;
42
```

## Config BD

```
JS db.js > client
1 // backend/db.js
2 const cassandra = require('cassandra-driver');
3
4 const client = new cassandra.Client({
5   contactPoints: ['127.0.0.1'],
6   localDataCenter: 'datacenter1',
7   keyspace: 'hotel_management',
8 });
9
10 module.exports = client;
11
```

### 1) Difficultés rencontrées et solutions

Pendant le développement du projet, plusieurs problèmes techniques ont été rencontrés :

- **Erreur de connexion entre frontend et backend** (ECONNREFUSED) : résolue en démarrant le backend Express avant le frontend React.
- **Backend manquant (server.js)** : solutionnée en créant un serveur Node.js de base avec Express.
- **Problèmes de configuration Cassandra** : évités grâce à l'utilisation de Docker pour exécuter Cassandra dans un conteneur.
- **Mauvais mapping des routes API** : corrigé en structurant les routes Express et en testant chaque endpoint avec Postman.
- **Affichage non responsive** : amélioré en utilisant Bootstrap pour une mise en page adaptable.

## VII) Présentation de Cassandra

### 1) Pourquoi Cassandra ?

Le choix de **Cassandra** comme base de données pour ce projet repose sur plusieurs critères techniques et fonctionnels adaptés aux exigences d'un système de gestion hôtelière à grande échelle.

Tout d'abord, **Cassandra est conçu pour gérer de très grands volumes de données** répartis sur plusieurs nœuds. Dans le contexte hôtelier, chaque réservation, paiement, facture ou interaction client génère des données qui doivent être stockées de manière efficace et consultables rapidement. Grâce à son architecture **distribuée et sans maître (masterless)**, Cassandra garantit une répartition automatique des données, ce qui permet un fonctionnement fluide et équilibré, sans point de défaillance unique.

L'un des avantages majeurs de Cassandra est sa **haute disponibilité**. Même en cas de panne d'un ou plusieurs nœuds, le système reste accessible, ce qui est essentiel pour une plateforme hôtelière qui doit être opérationnelle 24h/24, 7j/7. De plus, la **tolérance aux pannes** est assurée par la réplication des données entre nœuds, avec une gestion flexible du facteur de réplication selon les besoins de redondance.

Cassandra est également reconnu pour sa **scalabilité horizontale**. Contrairement aux bases traditionnelles, il est possible d'ajouter des nœuds au cluster à tout moment, sans perturber le service. Cela permet au système de s'adapter à la croissance naturelle de l'activité (nouvelles structures, plus de clients, extension géographique) tout en maintenant des performances constantes.

Sur le plan des performances, Cassandra offre une excellente gestion des **écritures intensives**. Chaque opération est enregistrée rapidement, ce qui est crucial dans un système qui traite continuellement de nouvelles réservations, annulations, paiements ou modifications de séjour. L'architecture basée sur des **fichiers immuables (SSTables)** et des écritures séquentielles en mémoire (MemTables) rend ce processus extrêmement rapide et efficace.

Enfin, le **modèle de données flexible** de Cassandra permet d'optimiser la structure des tables selon les besoins métiers. Il est possible de modéliser les données pour répondre à des cas d'usage précis, comme la consultation des séjours par hôtel et par période, ou encore les paiements par client. Cela permet d'offrir des réponses rapides à des requêtes fréquentes, tout en minimisant la complexité du traitement.

En résumé, Cassandra est un choix stratégique pour ce projet car il combine **performance, résilience, évolutivité et adaptabilité** à des environnements complexes à forte volumétrie de données comme celui de l'hôtellerie.

## 2) Origine de Cassandra

Cassandra est une base de données NOSQL (Not Only SQL) orientée colonne.

- Cassandra est développé par Facebook en 2007 la messagerie interne. En 2008, le projet est cédé à la fondation Apache et devient "top-level-project" à partir de 2010. Cassandra est alors enrichie et de nouvelles fonctionnalités y sont ajoutées.
- Cassandra est un projet open source.
- Cassandra est, entre autre, utilisée par Twitter, NetFlix ou Cisco WebEx.

## 3) Remarque: Les concepts de Cassandra

Cassandra reprend les concepts de 2 bases de données existantes :

- BigTable, créé par Google, pour son modèle de données orienté colonne et son mécanisme de persistance sur disque
- Dynamo, créé par Amazon, pour son architecture distribuée sans noeud maître.

## 4) Avantages / Inconvénients

### a) Avantages :

- Tolérance aux pannes (grâce aux mécanismes de répllication de données)
- Décentralisé
- Élastique
- Haute disponibilité
- Open Source

### b) Inconvénients :

- Lenteur
- Pas d'interface graphique
- Difficultés à l'utilisation
- Limitation de la taille des données

## VIII) Architecture de Cassandra

### 1) Définition: Cluster

Un cluster est un regroupement de plusieurs nœuds (serveur physique) qui communiquent entre eux pour la gestion de données.

### 2) Fondamental:

Cassandra est une base de données contenue dans un cluster.

Comme de nombreuses bases NoSQL, les données sont réparties sur plusieurs nœuds et peuvent être répliquées sur 1 à N nœuds.

Un utilisateur peut se connecter sur n'importe quel nœud et accéder à l'ensemble des données.

### 3) Colonne

La colonne est la plus petite unité de donnée de Cassandra. Elle est constituée du triplet :

- Nom
- Valeur : Maximum 2G. Elle n'est pas obligatoire.
- Timestamp : Sauvegarde la mise à jour la plus récente.

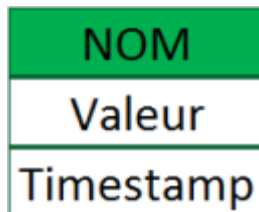


Figure 2: Colonne

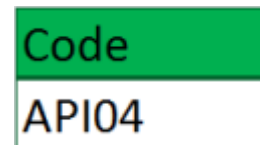


Figure 3: Exemple colonne

#### a) Remarque:

Une colonne dans Cassandra n'a pas le même sens que dans un SGBDR. Il s'agit là d'un attribut d'un enregistrement.

### 4) Définition: Ligne

Une ligne est un ensemble de colonnes (jusqu'à 2 Milliards). Elle est identifiée par une clé.

C'est l'équivalent d'une ligne dans un SGBDR, c'est donc un enregistrement.

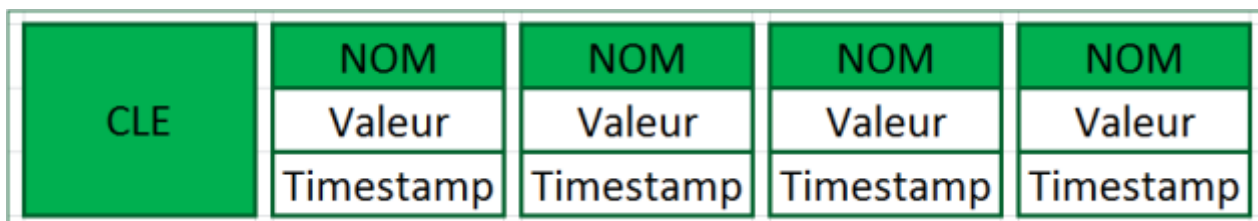


Figure 4: Ligne

CLE	Nom	Prénom	Age	Taille
	Moulin	Mathieu	16	180

Figure 5: Exemple de ligne

#### a) Remarque: La notion de clé dans Cassandra

La clé d'une ligne peut être rapprochée aux OID des SGBD RO. C'est grâce à elle que l'enregistrement est trouvé.

Les données étant réparties sur différents nœuds, il faut pouvoir la récupérer rapidement. Elle est codée sur 128 bits et est généralement représentée sous la forme de groupes de caractères hexadécimaux en minuscule séparés par des tirets. Cette clé d'enregistrement est de type uuid (Universal Unique Identifier).

Il est possible de définir des clés primaires similaires aux SGBDR. Dans ce cas, la première colonne de la clé sera utilisée comme clé d'enregistrement (et convertie). Les suivantes ne serviront qu'à s'assurer que l'enregistrement est unique.

#### 5) Définition: Column Family

Une Column Family est un regroupement de lignes.

Column Family				
CLE	NOM	NOM	NOM	NOM
	Valeur	Valeur	Valeur	Valeur
	Timestamp	Timestamp	Timestamp	Timestamp
CLE	NOM	NOM	NOM	NOM
	Valeur	Valeur	Valeur	Valeur
	Timestamp	Timestamp	Timestamp	Timestamp
CLE	NOM	NOM	NOM	NOM
	Valeur	Valeur	Valeur	Valeur
	Timestamp	Timestamp	Timestamp	Timestamp

Figure 6: Column family

Etudiants				
CLE	Nom	Prénom	Age	Taille
	Moulin	Mathieu	16	180
CLE	Nom	Prénom	Age	
	Dupond	Jean	23	
CLE	Nom	Prénom	Age	Taille
	Martin	Simon	22	175

Figure 7: Exemple de column family

**a) Remarque:**

Une Column Family est l'équivalent d'une Table dans un SGBDR.

On peut d'ailleurs y ajouter des "métadonnées", en quelque sorte des "entêtes" de colonnes. Néanmoins, les colonnes définies ne seront pas forcément exploitées lors de la création de ligne.

Il existe deux types de familles de colonnes

- statique : les colonnes sont définies lors de la création ou modification de la famille de colonnes ;
- dynamique : les colonnes sont définies lors de la création ou modification d'une ligne.

## 6) Définition: Keyspace

Un KeySpace est un regroupement de Column Family. Il équivaut au schéma dans un SGBDR.

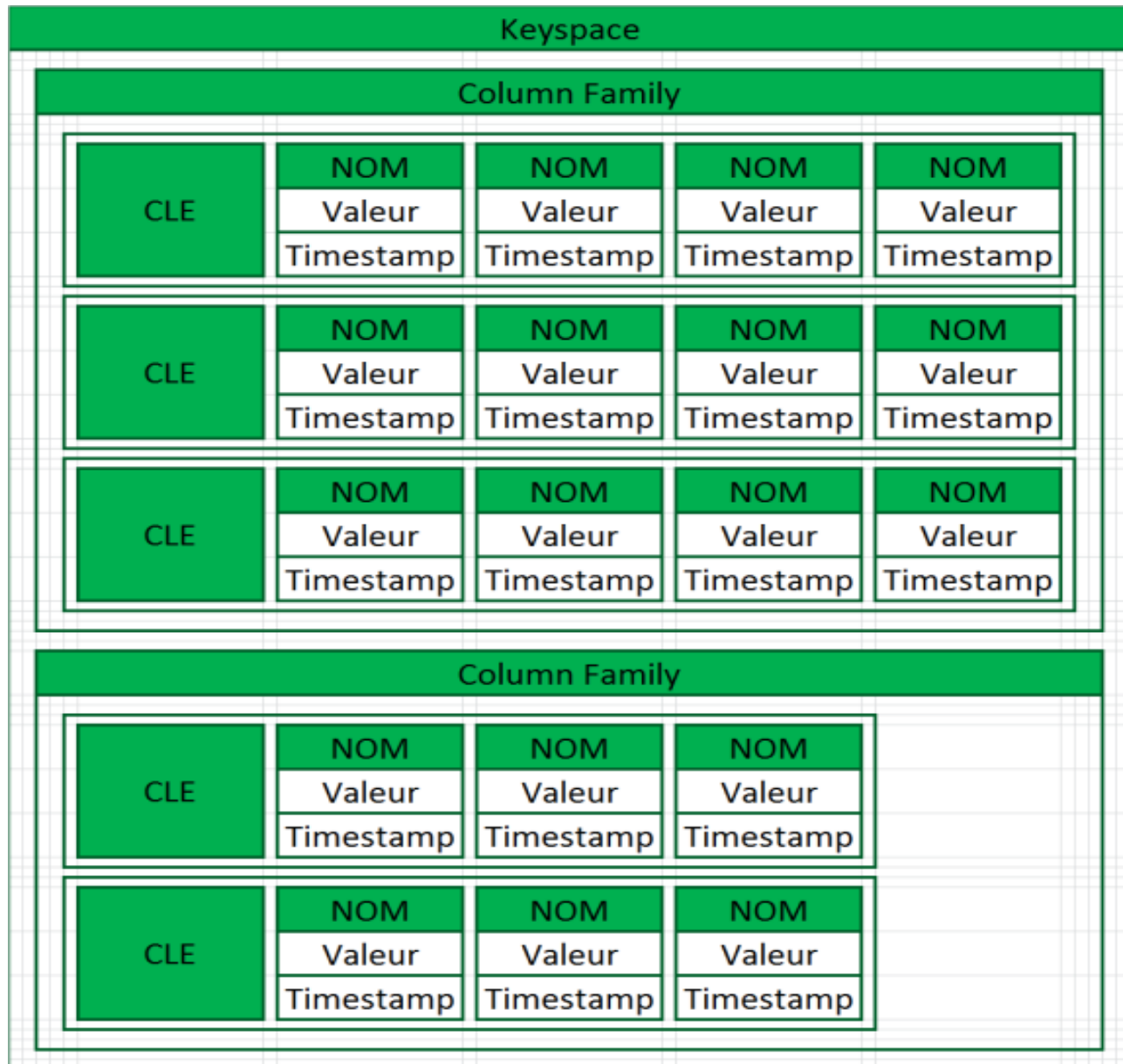


Figure 8: Keyspace



## IX) Gestion des données temporelles et financières

La gestion des données temporelles et financières représente un enjeu central dans le secteur hôtelier. Les systèmes doivent pouvoir enregistrer, analyser et exploiter efficacement l'ensemble des événements associés aux séjours, aux transactions et aux services. Dans ce projet, l'utilisation de Cassandra permet de répondre aux exigences de performance et de scalabilité inhérentes à ces types de données.

### 1) Historique des séjours et transactions financières

Dans le contexte d'un système de gestion hôtelière, il est indispensable de conserver un **historique complet des séjours** de chaque client, ainsi que l'ensemble des **transactions financières** associées. Les données collectées sont hétérogènes et comprennent notamment :

- **Les séjours** : identifiant du client, identifiant de l'hôtel, dates d'arrivée et de départ, type de chambre, services utilisés, nombre de nuits, statut de réservation (confirmée, annulée, terminée).
- **Les paiements** : montant, date de paiement, mode de règlement (carte, espèces, en ligne), statut (payé, en attente, rejeté), type de service (hébergement, restauration, spa...).
- **Les factures** : identifiant unique, liste des lignes facturées, total HT/TTC, TVA, référence du séjour associé, horodatage de génération.

L'ensemble de ces données sont **horodatées**, ce qui est fondamental pour :

- **Reconstituer le parcours du client** au sein d'un ou plusieurs établissements.
- **Produire des rapports temporels** : taux d'occupation par semaine/mois, chiffre d'affaires quotidien, pics de fréquentation.
- **Identifier des comportements clients** récurrents ou saisonniers pour orienter les stratégies marketing.
- **Assurer la traçabilité comptable** des opérations, ce qui est crucial pour les audits financiers ou les obligations fiscales.

Dans Cassandra, la structuration temporelle des données doit être soigneusement pensée pour éviter les lectures coûteuses ou les partitions trop volumineuses. Chaque ligne correspond généralement à un événement (paiement, séjour), et l'horodatage permet de filtrer ou d'agréger facilement ces données.

### 2) Optimisation des performances et scalabilité

L'un des principaux avantages de Cassandra est sa capacité à **gérer efficacement des volumes massifs de données**, tout en assurant une **réponse rapide et stable**, même en situation de forte charge. Cela en fait une solution parfaitement adaptée à la gestion des données temporelles dans un système hôtelier.

### 3) Stratégies de modélisation temporelle dans Cassandra

La base Cassandra n'est pas conçue pour effectuer des jointures complexes comme dans les bases relationnelles. Il est donc recommandé de **dupliquer certaines données** et de les structurer selon des cas d'usage bien définis. Par exemple :

- Une table `sejours_par_hotel_et_date` permet de retrouver tous les séjours enregistrés pour un hôtel donné sur une période précise.

- Une autre table paiements\_par\_client\_et\_période facilite le suivi des transactions par client selon les mois ou les trimestres.

Ces modèles permettent de **partitionner les données par période (jour, mois, année)** et/ou par identifiant d'hôtel ou de client. Cela limite la taille des partitions et optimise les temps de réponse.

#### 4) Utilisation des clés composites

Les **clés composites** dans Cassandra (clé primaire composée de plusieurs colonnes) sont essentielles pour organiser les données temporelles de manière efficace. Par exemple :

PRIMARY KEY ((id_hotel, mois), date_sejour)
---

- Ici, id\_hotel et mois forment la clé de partition.
- date\_sejour est la clé de clustering, ce qui permet de **trier les lignes dans l'ordre chronologique** à l'intérieur de chaque partition.

Ce modèle permet de :

- Lire rapidement les séjours d'un hôtel pour un mois donné.
- Parcourir les enregistrements dans l'ordre temporel sans coût supplémentaire.
- Éviter les partitions trop larges (en les regroupant par mois au lieu de stocker toutes les données dans une seule par hôtel).

#### 5) Scalabilité horizontale

Cassandra est conçue pour **scaler horizontalement**, c'est-à-dire qu'il est possible d'ajouter facilement de nouveaux nœuds au cluster pour **augmenter la capacité de stockage et la puissance de traitement**. Cette caractéristique est très importante lorsque le volume de données croît (ex : ajout de nouveaux hôtels, augmentation du nombre de réservations, enregistrement d'événements historiques).

La répartition des données entre les nœuds est automatique, grâce au mécanisme de **partitionnement par hachage**. Cela garantit une distribution équilibrée de la charge, sans point de défaillance unique. De plus, Cassandra gère la **réplication des données**, assurant leur disponibilité même en cas de panne d'un ou plusieurs nœuds.

### X) Fonctionnalités principales du système

L'objectif principal de ce projet est de concevoir un système capable de gérer efficacement les séjours, les paiements et l'historique des clients dans un environnement hôtelier. Grâce à la puissance de Cassandra et à une architecture bien pensée, plusieurs fonctionnalités clés ont été mises en place pour répondre aux besoins métiers en termes de gestion, de traçabilité et d'analyse.

## 2) Sauvegarde de l'historique des séjours par hôtel et période

Chaque séjour client doit être enregistré avec précision, en incluant les éléments suivants :

- Identifiant du client
- Hôtel concerné
- Date d'arrivée et de départ
- Type de chambre réservée
- Statut du séjour (en cours, terminé, annulé)

Pour optimiser l'accès à ces informations, une table a été modélisée dans Cassandra en tenant compte des axes **temporel** et **géographique** :

```
CREATE TABLE sejours_par_hotel (  
  id_hotel UUID,  
  mois TEXT,  
  date_sejour DATE,  
  id_client UUID,  
  nom_client TEXT,  
  chambre TEXT,  
  duree INT,  
  PRIMARY KEY ((id_hotel, mois), date_sejour)  
);
```

Ce modèle permet d'interroger facilement les séjours effectués dans un hôtel donné pendant une période précise (mois), tout en garantissant un tri chronologique grâce à la clé de clustering date\_sejour.

## 3) Gestion des paiements clients par période et type de service

La gestion des paiements est essentielle dans tout système de réservation. Chaque transaction doit être enregistrée avec les métadonnées nécessaires :

- Date et heure du paiement
- Montant
- Type de service (hébergement, restauration, spa...)
- Mode de paiement
- Statut du paiement (réussi, en attente, échoué)

Exemple de table Cassandra pour structurer les paiements par période :

```
CREATE TABLE paiements_par_pperiode (  
  periode TEXT,  
  id_client UUID,  
  id_hotel UUID,  
  date_paiement TIMESTAMP,  
  type_service TEXT,  
  montant DECIMAL,  
  mode_paiement TEXT,  
  PRIMARY KEY ((periode), date_paiement, id_client));
```

Cette organisation permet :

- D'effectuer des **rapports financiers mensuels ou hebdomadaires**
- De suivre l'historique de paiements par client ou par service
- D'agréger facilement les recettes d'un hôtel selon les périodes

#### 4) Stockage et analyse des données historiques

Toutes les données collectées (séjours, paiements, factures) sont conservées à des fins :

- De **reporting stratégique** (taux d'occupation, revenus par hôtel)
- D'**analyse marketing** (comportement des clients, pics saisonniers)
- De **conformité réglementaire** (archivage, fiscalité)

Pour cela, des **tables analytiques** supplémentaires peuvent être créées dans Cassandra, souvent redondantes mais optimisées pour des accès spécifiques. Par exemple :

```
CREATE TABLE taux_occupation_mensuel (  
  id_hotel UUID,  
  mois TEXT,  
  total_nuits INT,  
  chambres_disponibles INT,  
  taux_occupation DECIMAL,  
  PRIMARY KEY ((id_hotel), mois)  
);
```

Cette table facilite la production rapide de **statistiques de performance** hôtelière et l'export de données vers des outils de Business Intelligence (Power BI, Tableau...).

## **XI) Avantages et perspectives**

L'utilisation de Cassandra dans le développement de ce système de gestion hôtelière offre de nombreux avantages, tant sur le plan technique que fonctionnel. Ce choix s'inscrit dans une vision évolutive du projet, qui vise à répondre aux besoins actuels tout en anticipant les futurs enjeux liés à l'expansion, à l'analyse avancée des données, et à l'intégration d'autres services.

### **1) Avantages de l'approche choisie**

L'architecture et les technologies adoptées présentent plusieurs points forts :

#### **a) Scalabilité horizontale**

Cassandra permet d'ajouter des nœuds à la volée, sans interruption du service. Cela assure une évolution fluide de l'infrastructure avec la croissance des données (nouveaux hôtels, plus de clients, historique enrichi...).

#### **b) Haute disponibilité**

Grâce à son modèle distribué sans maître, le système reste opérationnel même si un ou plusieurs nœuds tombent en panne. Cela garantit une continuité de service, essentielle pour les plateformes accessibles 24h/24.

#### **c) Performance sur les écritures**

Les écritures sont rapides et efficaces, ce qui est idéal dans un contexte où les enregistrements (réservations, paiements, logs) sont fréquents.

#### **d) Modèle flexible**

La structure de Cassandra permet de modéliser les données selon les cas d'usage métier. Cela facilite l'accès aux données pertinentes sans requêtes complexes, tout en maintenant une bonne organisation.

#### **e) Adaptation aux données temporelles**

Cassandra gère très bien les données horodatées grâce à l'utilisation de clés composites. Cela permet de réaliser des analyses par période (jour, mois, année) avec des performances stables.

### **2) Possibilités d'extensions et analyses avancées**

L'architecture mise en place est conçue pour être **évolutive**. Plusieurs pistes peuvent être envisagées pour enrichir le système dans un futur proche :

#### **a) Ajout de modules fonctionnels**

- Gestion des avis clients.
- Module de fidélité.
- Suivi du personnel et planification des équipes.
- Intégration de services annexes : spa, restauration, événements.

### c) Analyse prédictive

En couplant Cassandra à des outils d'analyse avancée ou de machine learning, il est possible de :

- Prédire les périodes de forte affluence.
- Détecter des profils types de clients.
- Optimiser la tarification dynamique selon la demande.

### d) Tableaux de bord dynamiques

L'intégration avec des outils de BI comme **Power BI**, **Grafana** ou **Tableau** permettrait de construire des dashboards personnalisés :

- Taux d'occupation en temps réel.
- Chiffre d'affaires par hôtel ou service.
- Répartition des réservations par canal (web, agence, téléphone...).

### e) Vision future du projet

L'objectif à long terme est de faire évoluer ce système vers une **plateforme complète et intelligente**, intégrant :

- Une **gestion centralisée multi-hôtels** avec administration globale.
- Un hébergement **multi-cloud** pour plus de flexibilité.
- Un moteur de **recommandation personnalisée** pour les clients.

Une API ouverte permettant l'interconnexion avec d'autres systèmes (applications mobiles, partenaires touristiques...).

## XII) Expérimentation et discussions des résultats

### 1) La page d'accueil

La page d'accueil Au lancement de l'application, une fenêtre d'accueil s'affiche.

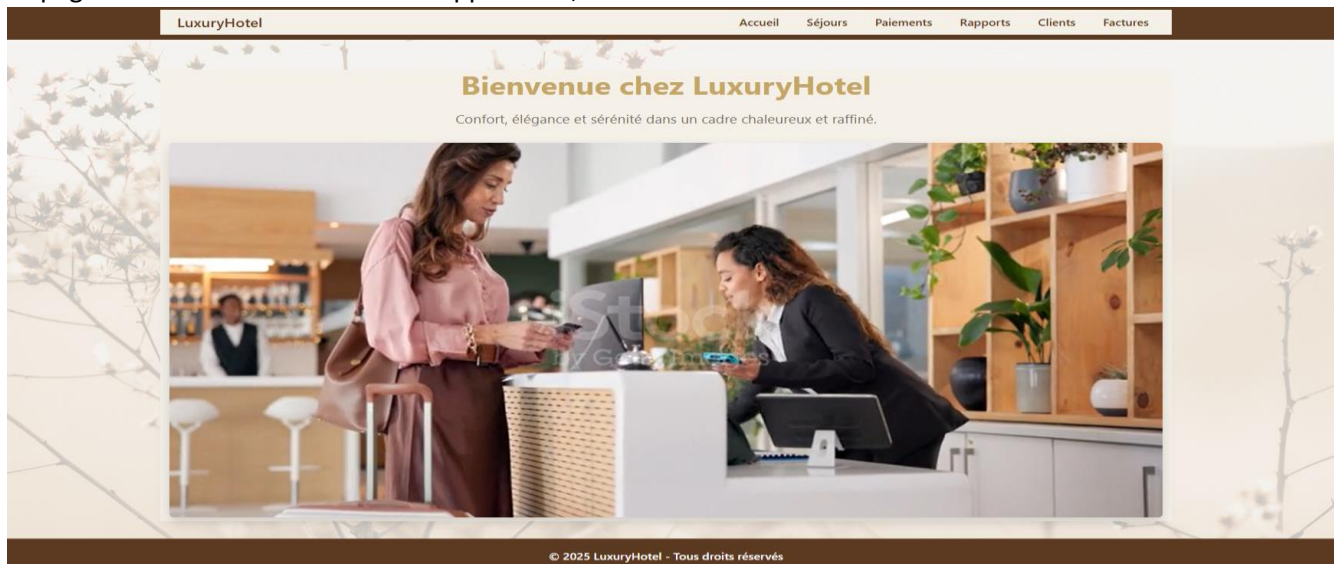


Figure 9: La page d'accueil

## 2) Clients





















Gestion client avec tous les options de CRUD + rechercher par nom/prénom.

### Liste des Clients

Ajouter un client

Ajouter

Rechercher

Nom	Prénom	Email	Téléphone	Actions
El Mansouri	Zakaria	zakaria.el mansouri@inwi.ma	0649586169	 
Naciri	Hind	hind.naciri@marocmail.ma	0675186843	 
Kabbaj	Omar	omar.kabbaj@gmail.com	0690569097	 
Zahidi	Youssef	youssef.zahidi@iam.ma	0620279740	 
Alaoui	Omar	omar.alaoui@gmail.com	0648020898	 
El Fassi	Zakaria	zakaria.el fassi@inwi.ma	0661506624	 
Berrada	Hind	hind.berrada@marocmail.ma	0629678041	 
Naciri	Zakaria	zakaria.naciri@inwi.ma	0645771799	 
El Mansouri	Nour	nour.el mansouri@iam.ma	0615374771	 
youssef	mansour	hindelmouden52@gmail.com	0696365280	 

Précédent

1

2

3

4

Suivant

Contrôle de validité des données saisies avant toute opération.

### Liste des Clients



Ajouter un client

Ajouter

Please fill out this field.

Figure 10: clients

## 3) Pagination

Naciri	Zakaria	zakaria.naciri@inwi.ma	0645771799	 
El Mansouri	Nour	nour.el mansouri@iam.ma	0615374771	 
youssef	mansour	hindelmouden52@gmail.com	0696365280	 

Précédent

1

2

3

4

Suivant

© 2025 LuxuryHotel - Tous droits réservés

Figure 11: pagination



## 5) Gestion des factures

LuxuryHotel

AccueilSéjoursPaielementsRapportsClientsFactures

Liste des Factures

Choisir un client

Montant Total

mm/dd/yyyy

Choisir un séjour

Type de facture

Ajouter

Rechercher par nom client

mm/dd/yyyy

mm/dd/yyyy

Client	Montant	Date Facture	Séjour	Actions
Omar Kabbaj	455.57	7/10/2025	excursion (7/10/2025) → (7/15/2025)	<div>Modifier</div> <div>Supprimer</div>
Omar Alaoui	407.08	5/20/2025	hébergement (5/19/2025) → (5/28/2025)	<div>Modifier</div> <div>Supprimer</div>
Hind Berrada	1415.52	3/29/2025	spa (3/28/2025) → (3/30/2025)	<div>Modifier</div> <div>Supprimer</div>
Hind Berrada	1590.8	3/28/2025	room-service (3/28/2025) → (3/30/2025)	<div>Modifier</div> <div>Supprimer</div>
Imane El Mansouri	1744.15	4/10/2025	room-service (4/10/2025) → (4/12/2025)	<div>Modifier</div> <div>Supprimer</div>
Youssef El Mansouri	991.63	7/3/2025	excursion (7/3/2025) → (7/5/2025)	<div>Modifier</div> <div>Supprimer</div>
Youssef El Fassi	1131.18	6/6/2025	excursion (6/6/2025) → (6/8/2025)	<div>Modifier</div> <div>Supprimer</div>
Mohamed El Mansouri	987.52	5/30/2025	restaurant (5/30/2025) → (6/7/2025)	<div>Modifier</div> <div>Supprimer</div>
Ayoub Alaoui	1062.27	6/7/2025	excursion (6/7/2025) → (6/16/2025)	<div>Modifier</div> <div>Supprimer</div>
Nour El Mansouri	918.3	6/8/2025	spa (6/8/2025) → (6/14/2025)	<div>Modifier</div> <div>Supprimer</div>

Précédent

1

2

3

4

Suivant

Figure 12: gestion des factures

## 6) Rapports statistiques

LuxuryHotel

AccueilSéjoursPaielementsRapportsClientsFactures

Rapports statistiques

Hotel ID

mm/dd/yyyy

mm/dd/yyyy

Taux d'occupation

Chiffre d'affaires

Total séjours

Paielements par type

Figure 13: Rapports statistiques



## 7) L'ajout d'un nouveau paiement

LuxuryHotel

AccueilSéjoursPalementsRapportsClientsFactures

Gestion des paiements

Annuler

Client ID

Montant

Mode (carte, cash...)

Description

Ajouter

797.69 MAD

Client: Youssef El Fassi

Mode: Virement

Date: 6/7/2025, 1:00:00 AM

Description: Paiement par Virement pour excursion

1119.9 MAD

Client: Mohamed El Mansouri

Mode: CB

Date: 5/9/2025, 1:00:00 AM

Description: Paiement par CB pour room-service

409.13 MAD

Client: Hind Naciri

Mode: Virement

Date: 6/19/2025, 1:00:00 AM

Description: Paiement par Virement pour restaurant

Figure 14: l'ajout d'un nouveau paiement

## 8) Gestion des paiements

LuxuryHotel

AccueilSéjoursPalementsRapportsClientsFactures

Gestion des paiements

+ Ajouter un paiement

797.69 MAD

Client: Youssef El Fassi

Mode: Virement

Date: 6/7/2025, 1:00:00 AM

Description: Paiement par Virement pour excursion

1119.9 MAD

Client: Mohamed El Mansouri

Mode: CB

Date: 5/9/2025, 1:00:00 AM

Description: Paiement par CB pour room-service

409.13 MAD

Client: Hind Naciri

Mode: Virement

Date: 6/19/2025, 1:00:00 AM

Description: Paiement par Virement pour restaurant

393.71 MAD

Client: Ayoub Kabbaj

Mode: Espèces

Date: 6/4/2025, 1:00:00 AM

Description: Paiement par Espèces pour excursion

906.68 MAD

Client: Siham Naciri

Mode: Espèces

Date: 6/17/2025, 1:00:00 AM

Description: Paiement par Espèces pour hébergement

687.01 MAD

Client: Ayoub Alaoui

Mode: Espèces

Date: 6/8/2025, 1:00:00 AM

Description: Paiement par Espèces pour hébergement

Figure 15: Gestion des paiements

## 9) Gestion des Séjours

Gestion des Séjours avec toutes les options de CRUD + filtrage par date ou ID

### Gestion des Séjours

Client ID	Hotel ID	Date début	Date fin	Montant	Actions
71858222-5115-45d9-b2ce-c8f3e3a81069	297d87d9-5594-4f70-a90d-37152e2ab6a3	5/10/2025	5/18/2025	1282.20 MAD	
2f719288-95f2-47d8-9bd3-f2844d0aa58a	038223f4-c618-47c0-8fe0-829ec1a4f894	6/6/2025	6/8/2025	252.50 MAD	
5490fe1a-8d22-4190-9ead-87e4a217699e	75293c3a-dbbe-4b2e-a55d-26359bceade3	7/10/2025	7/15/2025	718.62 MAD	
e1c48149-57a9-419c-a88b-a66c20e07f57	b1b1501b-7143-40a6-b8c1-948617d9255c	4/29/2025	5/8/2025	1316.54 MAD	
187ec10a-0545-420c-87af-ec3949313fba	9512a121-1784-482b-905f-6c01719e9bf0	5/28/2025	5/31/2025	580.10 MAD	
59e253d7-acbb-420d-ae7d-af34c8ac4108	13861879-5c49-44fe-9cde-e625ac6c9d1b	5/30/2025	6/7/2025	976.07 MAD	
db389b38-bbf2-498e-8fd2-c3f057293930	9e862071-8f5b-416b-ba79-a59c083201ed	6/25/2025	6/29/2025	735.87 MAD	
da0c761e-0f68-40a8-92da-af49946f07bf	17e22c48-2d37-4868-b660-f76017069b6	3/28/2025	3/30/2025	414.56 MAD	
62dde09f-f81a-4c04-9647-6baabf7a27fc	9c1aa65b-3c3a-4a65-ac7e-c08e92498dbd	6/7/2025	6/16/2025	1167.31 MAD	
542aa653-912f-406d-a27e-753fcb637038	682c0a8f-b220-441d-bc5d-be52f0faa103	6/4/2025	6/13/2025	1316.86 MAD	

1
2
3

Figure 16: Gestion des Séjours

Client	Montant	Date Facture	Séjour	Actions
Youssef El Mansouri	991.63	7/3/2025	excursion (7/3/2025) → (7/5/2025)	Modifier  Supprimer
Youssef El Fassi	1131.18	6/6/2025	excursion (6/6/2025) → (6/8/2025)	Modifier  Supprimer
Youssef El Fassi	786.75	5/28/2025	room-service (5/28/2025) → (5/31/2025)	Modifier  Supprimer
Youssef El Mansouri	982.49	7/4/2025	spa (7/3/2025) → (7/5/2025)	Modifier  Supprimer
Youssef El Fassi	1002.71	6/7/2025	room-service (6/6/2025) → (6/8/2025)	Modifier  Supprimer
Youssef Tazi	1604.14	7/20/2025	excursion (7/20/2025) → (7/25/2025)	Modifier  Supprimer
Youssef Kabbaj	1272.05	5/11/2025	spa (5/11/2025) → (5/20/2025)	Modifier  Supprimer
Youssef Zahidi	321.75	5/26/2025	room-service (5/26/2025) → (6/3/2025)	Modifier  Supprimer

Figure 17 : Recherche instantané

## 10) Table factures

```
cqlsh:hotel_management>
cqlsh:hotel_management> DESCRIBE TABLES;

clients      factures_by_client  paiements  stats_periode
factures     factures_by_sejour  sejours

cqlsh:hotel_management> DESCRIBE TABLE factures;

CREATE TABLE hotel_management.factures (
  id uuid PRIMARY KEY,
  client_id uuid,
  date_facture date,
  montant_total double,
  sejour_id uuid,
  type_facture text
) WITH additional_write_policy = '99p'
  AND allow_auto_snapshot = true
  AND bloom_filter_fp_chance = 0.01
  AND caching = {'keys': 'ALL', 'rows_per_partition': 'NONE'}
  AND cdc = false
  AND comment = ''
  AND compaction = {'class': 'org.apache.cassandra.db.compaction.SizeTieredCompactionStrategy', 'max_threshold': '32', 'min_threshold': '4'}
  AND compression = {'chunk_length_in_kb': '16', 'class': 'org.apache.cassandra.io.compress.LZ4Compressor'}
  AND memtable = 'default'
  AND crc_check_chance = 1.0
  AND default_time_to_live = 0
  AND extensions = {}
  AND gc_grace_seconds = 864000
  AND incremental_backups = true
  AND max_index_interval = 2048
  AND memtable_flush_period_in_ms = 0
  AND min_index_interval = 128
  AND read_repair = 'BLOCKING'
  AND speculative_retry = '99p';
cqlsh:hotel_management> DESCRIBE TABLE paiements;

CREATE TABLE hotel_management.paiements (
```

Figure 17: table factures

## 11) Table paiements

```
  AND speculative_retry = '99p';
cqlsh:hotel_management> DESCRIBE TABLE paiements;

CREATE TABLE hotel_management.paiements (
  id uuid PRIMARY KEY,
  client_id uuid,
  date timestamp,
  date_paiement date,
  description text,
  facture_id uuid,
  montant double,
  sejour_id uuid,
  type text
) WITH additional_write_policy = '99p'
  AND allow_auto_snapshot = true
  AND bloom_filter_fp_chance = 0.01
  AND caching = {'keys': 'ALL', 'rows_per_partition': 'NONE'}
  AND cdc = false
  AND comment = ''
  AND compaction = {'class': 'org.apache.cassandra.db.compaction.SizeTieredCompactionStrategy', 'max_thresl
  AND compression = {'chunk_length_in_kb': '16', 'class': 'org.apache.cassandra.io.compress.LZ4Compressor'}
  AND memtable = 'default'
  AND crc_check_chance = 1.0
  AND default_time_to_live = 0
  AND extensions = {}
  AND gc_grace_seconds = 864000
  AND incremental_backups = true
  AND max_index_interval = 2048
  AND memtable_flush_period_in_ms = 0
  AND min_index_interval = 128
  AND read_repair = 'BLOCKING'
  AND speculative_retry = '99p';
cqlsh:hotel_management>
```

Figure 18: Table paiements

## 12) Table clients

```

CREATE TABLE hotel_management.clients (
  id uuid PRIMARY KEY,
  email text,
  nom text,
  prenom text,
  telephone text,
  vip boolean
) WITH additional_write_policy = '99p'
  AND allow_auto_snapshot = true
  AND bloom_filter_fp_chance = 0.01
  AND caching = {'keys': 'ALL', 'rows_per_partition': 'NONE'}
  AND cdc = false
  AND comment = ''
  AND compaction = {'class': 'org.apache.cassandra.db.compaction.SizeTieredCompactionStrategy'}
  AND compression = {'chunk_length_in_kb': '16', 'class': 'org.apache.cassandra.io.compress.LZ4Compressor'}
  AND memtable = 'default'
  AND crc_check_chance = 1.0
  AND default_time_to_live = 0
  AND extensions = {}
  AND gc_grace_seconds = 864000
  AND incremental_backups = true
  AND max_index_interval = 2048
  AND memtable_flush_period_in_ms = 0
  AND min_index_interval = 128
  AND read_repair = 'BLOCKING'
  AND speculative_retry = '99p';

CREATE INDEX clients_nom_idx ON hotel_management.clients (nom);

```

Figure 19: Table clients

### 13) Table statuts période

```

cqlsh:hotel_management> DESCRIBE table stats_periode;

CREATE TABLE hotel_management.stats_periode (
  periode text PRIMARY KEY,
  total_montant double,
  total_sejours int
) WITH additional_write_policy = '99p'
  AND allow_auto_snapshot = true
  AND bloom_filter_fp_chance = 0.01
  AND caching = {'keys': 'ALL', 'rows_per_partition': 'NONE'}
  AND cdc = false
  AND comment = ''
  AND compaction = {'class': 'org.apache.cassandra.db.compaction.SizeTieredCompactionStrategy', 'max_threshold_in_kb': 256}
  AND compression = {'chunk_length_in_kb': '16', 'class': 'org.apache.cassandra.io.compress.LZ4Compressor'}
  AND memtable = 'default'
  AND crc_check_chance = 1.0
  AND default_time_to_live = 0
  AND extensions = {}
  AND gc_grace_seconds = 864000
  AND incremental_backups = true
  AND max_index_interval = 2048
  AND memtable_flush_period_in_ms = 0
  AND min_index_interval = 128
  AND read_repair = 'BLOCKING'
  AND speculative_retry = '99p';

```

Figure 20: statut périodes

## 14) Factures par client

```
cqlsh:hotel_management> DESCRIBE table factures_by_client ;

CREATE TABLE hotel_management.factures_by_client (
  client_id uuid,
  date_facture date,
  id uuid,
  montant_total double,
  sejour_id uuid,
  type_facture text,
  PRIMARY KEY (client_id, date_facture, id)
) WITH CLUSTERING ORDER BY (date_facture DESC, id ASC)
  AND additional_write_policy = '99p'
  AND allow_auto_snapshot = true
  AND bloom_filter_fp_chance = 0.01
  AND caching = {'keys': 'ALL', 'rows_per_partition': 'NONE'}
  AND cdc = false
  AND comment = ''
  AND compaction = {'class': 'org.apache.cassandra.db.compaction.SizeTieredCompactionStrategy', 'max_t
  AND compression = {'chunk_length_in_kb': '16', 'class': 'org.apache.cassandra.io.compress.LZ4Compress
  AND memtable = 'default'
  AND crc_check_chance = 1.0
  AND default_time_to_live = 0
  AND extensions = {}
  AND gc_grace_seconds = 864000
  AND incremental_backups = true
  AND max_index_interval = 2048
  AND memtable_flush_period_in_ms = 0
  AND min_index_interval = 128
  AND read_repair = 'BLOCKING'
  AND speculative_retry = '99p';
```

Figure 21: Factures par client

## 15) Facture par séjours

```
cqlsh:hotel_management> DESCRIBE table factures_by_sejour ;  
  
CREATE TABLE hotel_management.factures_by_sejour (  
    sejour_id uuid,  
    date_facture date,  
    id uuid,  
    client_id uuid,  
    montant_total double,  
    type_facture text,  
    PRIMARY KEY (sejour_id, date_facture, id)  
) WITH CLUSTERING ORDER BY (date_facture DESC, id ASC)  
    AND additional_write_policy = '99p'  
    AND allow_auto_snapshot = true  
    AND bloom_filter_fp_chance = 0.01  
    AND caching = {'keys': 'ALL', 'rows_per_partition': 'NONE'}  
    AND cdc = false  
    AND comment = ''  
    AND compaction = {'class': 'org.apache.cassandra.db.compaction.SizeTieredCo  
    AND compression = {'chunk_length_in_kb': '16', 'class': 'org.apache.cassand  
    AND memtable = 'default'  
    AND crc_check_chance = 1.0  
    AND default_time_to_live = 0  
    AND extensions = {}  
    AND gc_grace_seconds = 864000  
    AND incremental_backups = true  
    AND max_index_interval = 2048  
    AND memtable_flush_period_in_ms = 0  
    AND min_index_interval = 128
```

Figure 22: Facture par séjours



# Conclusion

Ce projet a permis la conception et la mise en œuvre d'un **système de gestion hôtelière moderne**, reposant sur la base de données NoSQL **Apache Cassandra**. En adoptant une approche axée sur la **performance**, la **disponibilité** et la **scalabilité**, nous avons su répondre aux exigences spécifiques du secteur hôtelier en matière de gestion de **donnée temporelles, financières et analytiques**.

L'étude approfondie de Cassandra a mis en lumière ses nombreux atouts : une **architecture distribuée et tolérante aux pannes**, une **excellente gestion des écritures**, ainsi qu'une **capacité éprouvée à traiter de très grands volumes de données**. La modélisation a été pensée pour épouser les usages réels du métier, offrant un accès optimisé aux informations clés telles que les séjours, les paiements, les factures et les indicateurs de performance.

L'intégration d'outils complémentaires tels que **Docker** (pour le déploiement), **React.js** et **Bootstrap** (pour l'interface utilisateur), ou encore **Node.js** et **Express** (pour l'API backend), a permis de bâtir une solution **fiable, évolutive et conviviale**, pensée pour offrir une expérience fluide aussi bien aux développeurs qu'aux utilisateurs finaux.

Au-delà des aspects purement techniques, ce projet constitue une **base solide pour des évolutions futures**: ajout de modules fonctionnels (avis, fidélisation, gestion du personnel), intégration d'outils d'**analyse prédictive** ou de **business intelligence**, ou encore **déploiements multisites** dans un environnement cloud.

Ces perspectives ouvrent la voie à un **système hôtelier intelligent, agile et centré sur l'expérience client**, reposant sur la robustesse de Cassandra pour bâtir les fondations d'une gestion numérique durable, efficace et tournée vers l'avenir.