



**Réalisé par :**

Imad ouairem

Mohamed El Amri

Anas Touloun

**Rapport Projet : Architecture des composants d’entreprise**

**Réalisation d’une application gestion réservation des salles**





**Ingénierie Informatique et Réseaux**

**Aperçu du projet**

**Introduction**

Le développement d'une application de réservation de salles de réunion est une réponse directe aux défis croissants auxquels font face les entreprises modernes en matière de gestion d'espace. Dans un monde où l'efficacité et la flexibilité sont primordiales, il est essentiel de disposer d'un système capable de gérer efficacement les ressources spatiales. Cette application vise à simplifier le processus de réservation, offrant aux utilisateurs une interface conviviale et une visibilité en temps réel sur la disponibilité des salles.

La nécessité d'une telle solution est renforcée par l'évolution rapide des environnements de travail. Les entreprises recherchent des solutions qui non seulement optimisent l'utilisation des espaces, mais qui favorisent également la collaboration et la flexibilité. Notre application s'inscrit dans cette tendance, offrant une plateforme intuitive qui facilite la planification des réunions et des événements, tout en s'adaptant aux besoins changeants des organisations.

**Problématique Générale**

La problématique générale de ce projet réside dans la gestion inefficace des salles de réunion, souvent marquée par des processus de réservation dépassés, une utilisation sous-optimale des espaces, et des conflits d'horaires fréquents. Ces problèmes engendrent non seulement une perte de temps et des frustrations parmi les employés, mais ils affectent également la productivité globale de l'entreprise. Il devient alors essentiel de développer une solution capable de gérer ces défis de manière efficace.

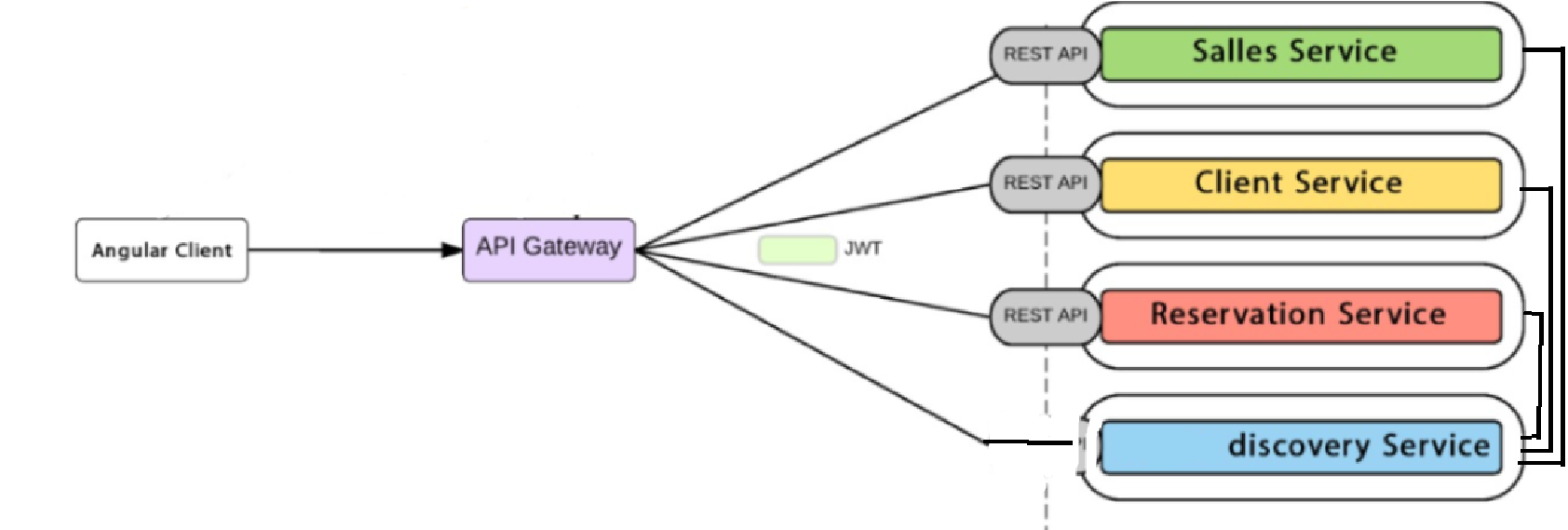
En outre, la problématique s'étend aux limitations des systèmes existants, qui peinent souvent à offrir la flexibilité et l'adaptabilité nécessaires dans un environnement professionnel dynamique. Les systèmes actuels ne parviennent pas à répondre aux besoins spécifiques des utilisateurs, manquant de fonctionnalités personnalisées et d'une intégration fluide avec d'autres outils de gestion d'entreprise. Notre projet vise à combler ces lacunes en introduisant une application de réservation innovante et polyvalente.

**Importance de l'architecture microservices**

L'architecture microservices est devenue de plus en plus importante dans le domaine du développement logiciel en raison de plusieurs avantages qu'elle offre par rapport à l'architecture monolithique traditionnelle. Voici quelques-unes des raisons pour lesquelles l'architecture microservices est considérée comme importante :

1. **Évolutivité et agilité :** Les microservices permettent une évolutivité plus facile. Chaque service peut être développé, déployé et mis à l'échelle indépendamment des autres. Cela permet aux équipes de développement d'être plus agiles, en pouvant mettre en œuvre des changements plus rapidement et de manière incrémentale.
2. **Isolation des services :** Chaque microservice est une entité indépendante qui peut être développée, déployée et évoluée sans affecter les autres services. Cela permet une meilleure isolation des erreurs et une gestion plus efficace des mises à jour.
3. **Flexibilité technologique :** Les microservices permettent l'utilisation de différentes technologies pour chaque service en fonction des besoins spécifiques. Cela permet aux équipes de choisir les meilleurs outils et technologies pour chaque composant, plutôt que d'être limitées par une seule technologie pour l'ensemble de l'application.
4. **Facilité de déploiement continu :** La nature indépendante des microservices facilite la mise en œuvre de stratégies de déploiement continu. Les mises à jour peuvent être effectuées service par service sans perturber l'ensemble de l'application.
5. **Évolutivité horizontale :** La conception des microservices facilite l'évolutivité horizontale, c'est-à-dire l'ajout de nouvelles instances de services pour gérer une charge accrue. Cela permet de faire face à des pics de demande sans avoir à augmenter la taille de l'ensemble de l'application.
6. **Meilleure résilience :** En cas de défaillance d'un microservice, les autres services peuvent continuer à fonctionner normalement. Cela améliore la résilience globale du système.
7. **Facilité de gestion :** Les équipes peuvent se spécialiser dans des domaines spécifiques en se concentrant sur le développement, le déploiement et la maintenance d'un ou plusieurs microservices plutôt que de gérer un monolithe complexe.
8. **Meilleure adaptation aux architectures de cloud :** Les architectures de microservices sont bien adaptées aux environnements de cloud computing. Elles peuvent tirer parti des fonctionnalités telles que l'élasticité, la mise en réseau distribuée et les services gérés proposés par les fournisseurs de cloud.

**Architecture Microservices**

****

**Description des services**

Notre application de réservation de salles de réunion est construite sur une architecture basée sur des microservices, offrant une modularité, une scalabilité et une maintenance facilitée. Chaque microservice joue un rôle spécifique dans l'application et fonctionne de manière indépendante, tout en s'intégrant harmonieusement au sein de l'architecture globale. Voici une présentation détaillée de nos principaux microservices :

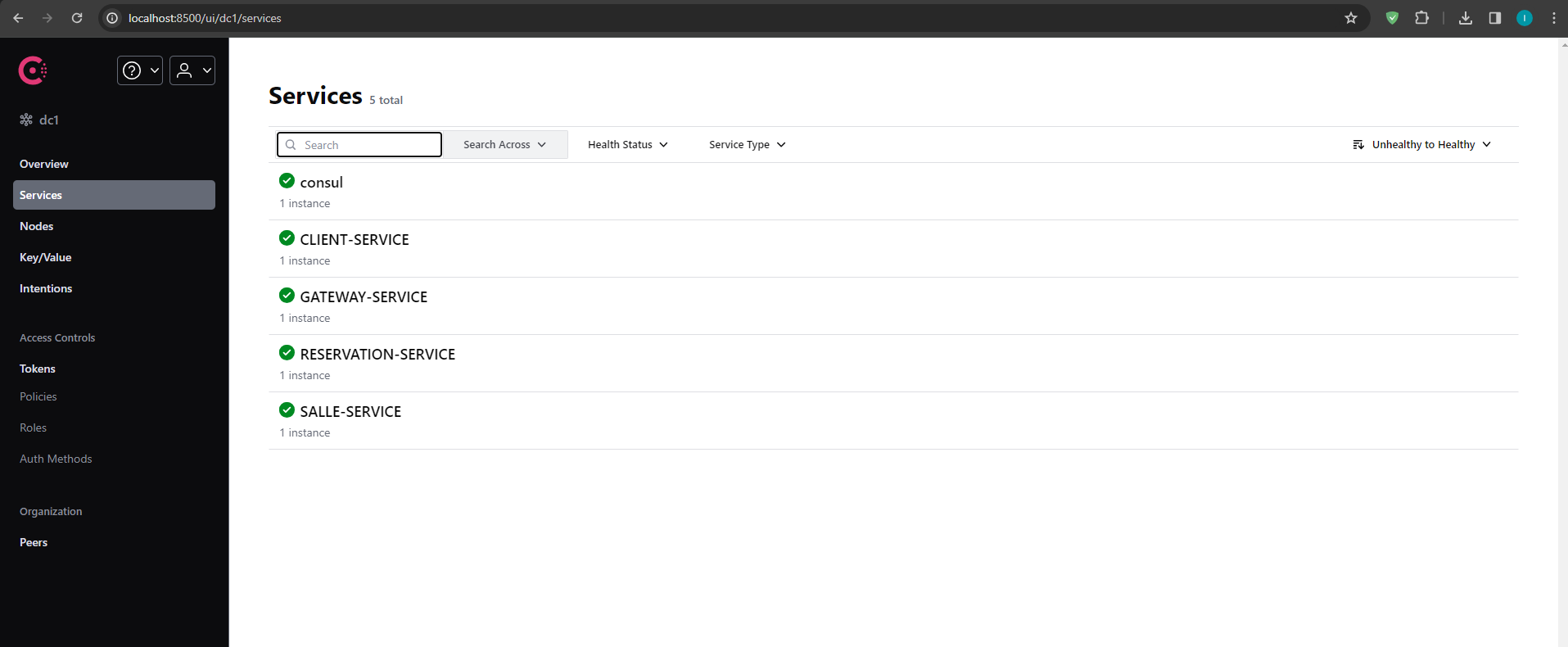
* **Consul** : Consul, qui est un logiciel de découverte de services et de gestion des configurations, souvent utilisé dans des architectures de microservices et de cloud.
* **Gateway Service :** Le Gateway Service sert de porte d'entrée à notre application, acheminant les requêtes entrantes vers les microservices appropriés. Il gère également les aspects de sécurité comme l'authentification et la protection contre les attaques. Ce service assure que les requêtes sont traitées efficacement et en toute sécurité.
* **Salle-Service :** Ce microservice est responsable de la gestion de toutes les informations relatives aux salles de réunion, incluant leur disponibilité, leurs caractéristiques et leur réservation. Il permet aux utilisateurs de visualiser et de sélectionner les salles en fonction de leurs besoins spécifiques.
* **Client-Service :** Le Client-Service gère les données utilisateur, telles que les informations de profil, les préférences et l'historique

de réservation. Il est essentiel pour offrir une expérience utilisateur personnalisée et pour maintenir les informations des clients à jour et sécurisées.

* **Reservation-Service** : Ce microservice est au cœur de l'application, traitant la logique et les fonctionnalités liées à la réservation des salles. Il gère les demandes de réservation, les modifications, les annulations et coordonne avec le Salle-Service pour assurer une gestion efficace des disponibilités de salles.

Chacun de ces microservices est conçu pour être autonome, ce qui permet une maintenance et une mise à jour simplifiées, ainsi qu'une meilleure gestion des ressources. Cette approche modulaire garantit que l'application est facilement scalable, pouvant s'adapter à une augmentation de la demande ou à l'ajout de nouvelles fonctionnalités.

**Consul :**



**Mécanismes de communication**

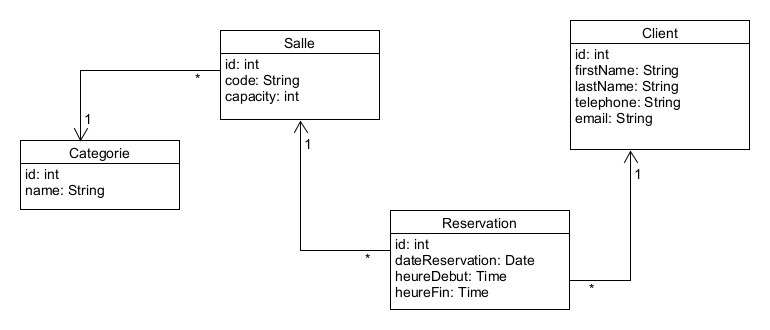
Dans l'architecture microservices décrite, les différents services, tels que Reservation-Service, la Salle-Service et le Client-Service, communiquent entre eux de manière efficace grâce à l'utilisation d'OpenFeign. OpenFeign est une bibliothèque déclarative de communication avec les services HTTP, développée par Netflix, et souvent utilisée dans les architectures basées sur les microservices.

Chaque microservice, comme la Salle-Service et le Client-Service, expose une interface API qui définit les opérations qu'il prend en charge. OpenFeign permet aux microservices de déclarer de manière propre et simple comment ils souhaitent interagir avec d'autres services. Plutôt que de gérer manuellement les appels HTTP, OpenFeign utilise des annotations et des interfaces pour définir les appels aux services distants de manière transparente.

Par exemple, le Reservation Service, lorsqu'il a besoin d'interagir avec la Salle-Service pour récupérer des informations sur la disponibilité des salles de réunion, utilise OpenFeign pour déclarer cette interaction. OpenFeign génère automatiquement le code nécessaire pour effectuer la requête HTTP, simplifiant ainsi le processus de communication entre les microservices.

L'utilisation d'OpenFeign facilite la gestion des appels distants, réduit la complexité du code et favorise une communication efficace entre les différents composants du système. Cela contribue à la flexibilité, à la scalabilité et à la facilité de maintenance de l'architecture microservices dans son ensemble, offrant ainsi une solution robuste pour le développement d'applications distribuées.

**Conception des Services**



**Conteneurisation avec Docker**

La conteneurisation avec Docker est une méthode populaire pour le déploiement d'applications, la gestion de l'infrastructure et la création d'environnements isolés et portables. Docker est une plateforme open source qui facilite la création, le déploiement et la gestion de conteneurs. Un conteneur est une unité logicielle légère qui encapsule une application et ses dépendances, permettant une exécution cohérente et reproductible sur différents environnements.

principaux concepts liés à la conteneurisation avec Docker et quelques avantages associés :

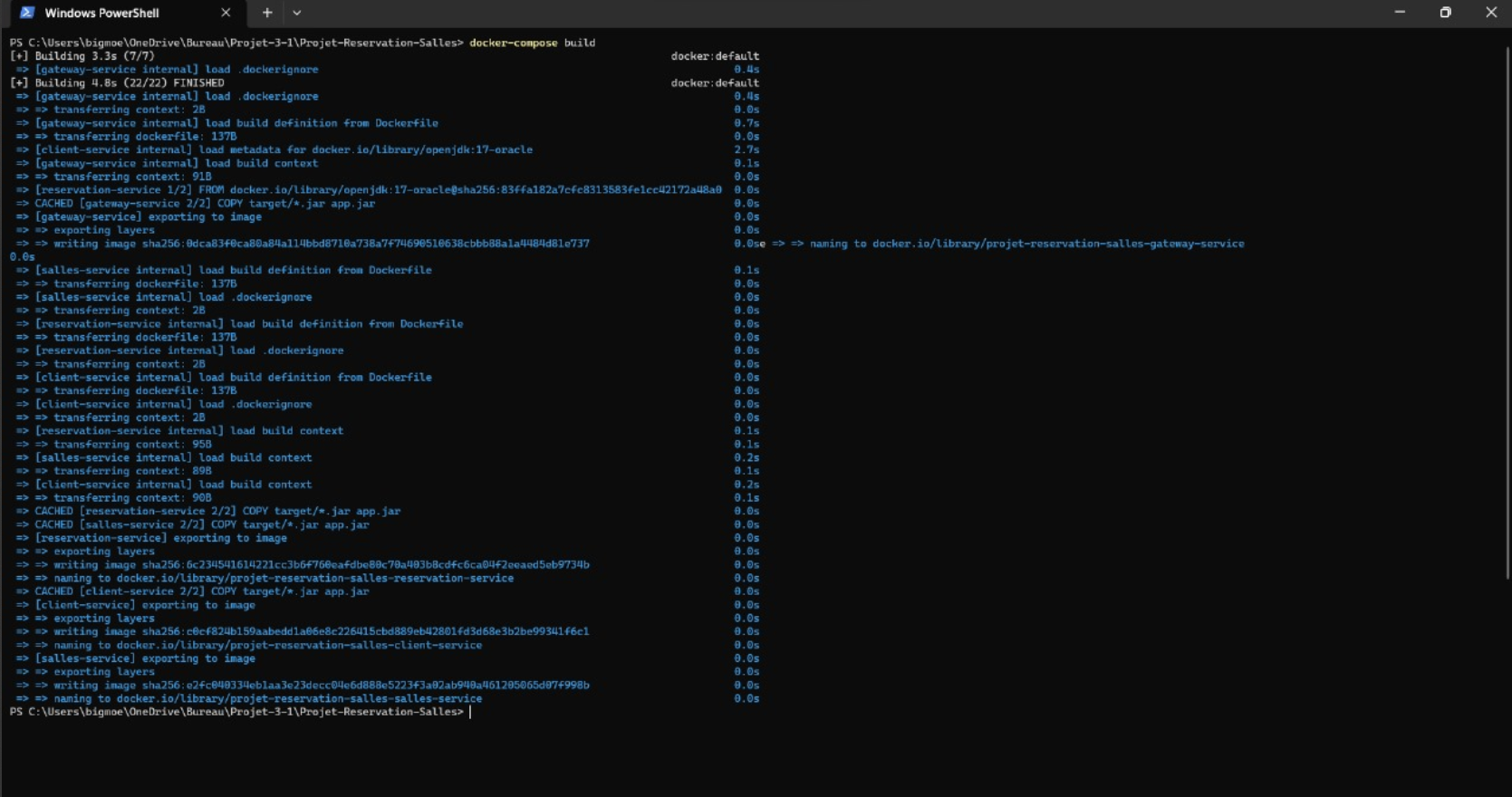
1. **Conteneurs :** Un conteneur est une instance exécutable d'une application, isolée du système d'exploitation hôte et des autres conteneurs. Il contient tout ce dont une application a besoin pour s'exécuter, y compris le code, les bibliothèques, les dépendances et les variables d'environnement.
2. **Images Docker :** Les images Docker sont des artefacts immuables qui servent de modèle pour créer des conteneurs. Elles contiennent l'ensemble des fichiers et configurations nécessaires à l'exécution d'une application. Les images peuvent être partagées via des registres Docker tels que Docker Hub.
3. **Dockerfile :** Un Dockerfile est un script qui définit les étapes nécessaires pour créer une image Docker. Il spécifie la configuration de l'environnement, les dépendances et les commandes d'exécution de l'application.

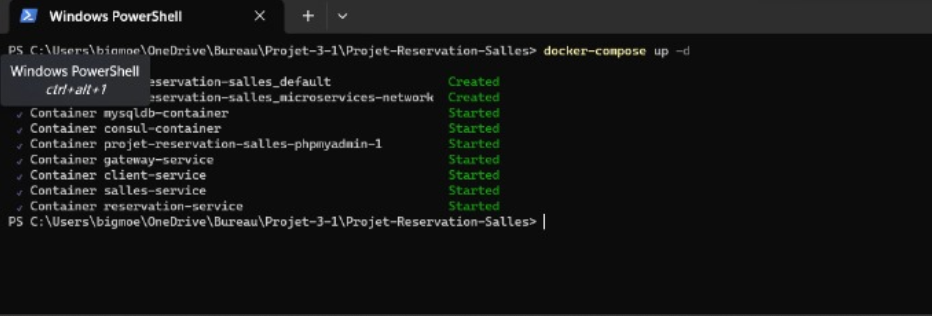
**Avantages de la conteneurisation avec Docker :**

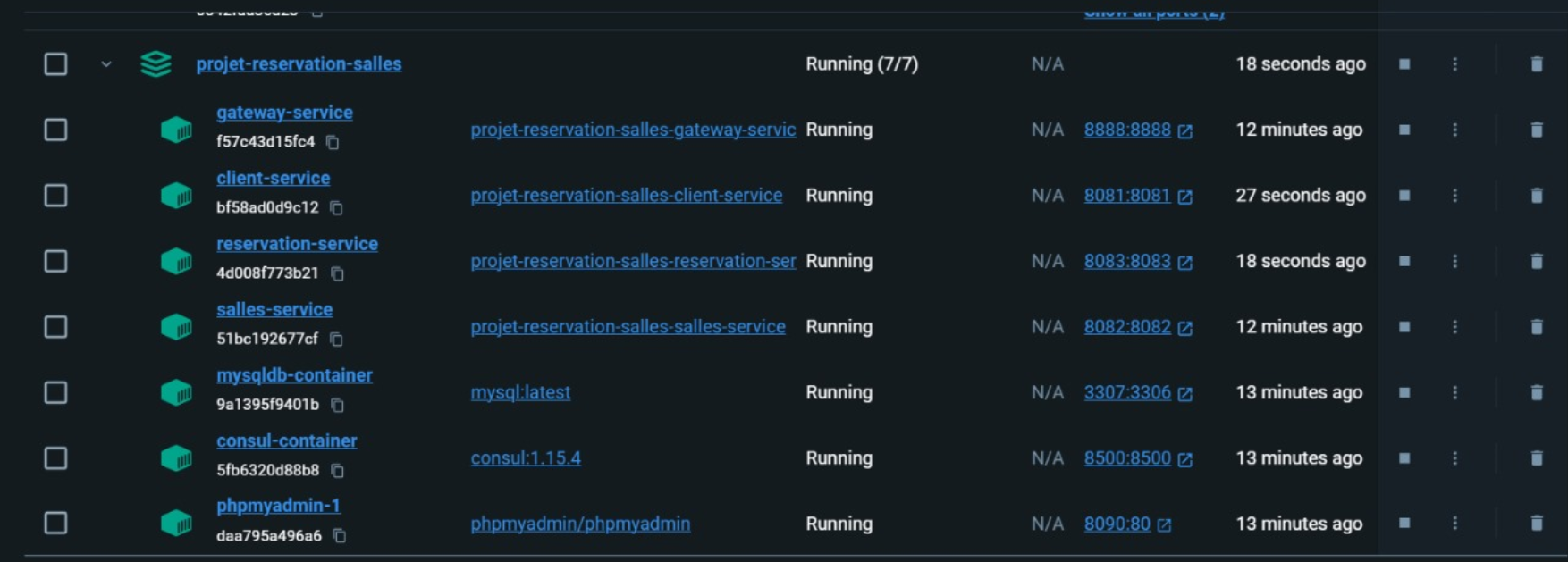
* **Portabilité :** Les conteneurs Docker encapsulent tout ce dont une application a besoin, garantissant ainsi une portabilité élevée. Un conteneur peut être exécuté de manière cohérente sur n'importe quel système prenant en charge Docker, qu'il s'agisse d'un environnement de développement, de test ou de production.
* **Isolation :** Les conteneurs offrent une isolation légère entre l'application et le système d'exploitation hôte. Cela évite les conflits entre les dépendances et facilite la gestion des ressources.
* **Rapidité de déploiement :** Les conteneurs peuvent être démarrés et arrêtés rapidement, permettant un déploiement rapide et des mises à l'échelle dynamiques en fonction de la charge.
* **Gestion des dépendances :** Les images Docker capturent toutes les dépendances nécessaires à l'exécution de l'application, éliminant ainsi les problèmes de compatibilité entre différentes versions de bibliothèques.
* **Gestion centralisée :** Docker propose des outils de gestion centralisée, facilitant la gestion des conteneurs à l'échelle, la mise à jour des applications et la surveillance.
* **Environnements reproductibles :** Grâce aux images Docker et aux Dockerfiles, les environnements de développement, de test et de production peuvent être reproduits de manière cohérente, réduisant ainsi les erreurs liées aux différences d'environnement.
* **Écosystème riche :** Docker dispose d'un vaste écosystème, avec une variété d'outils et de services complémentaires qui facilitent le développement, le déploiement et la gestion des conteneurs.

La conteneurisation avec Docker a transformé la façon dont les applications sont développées, déployées et gérées, offrant une solution efficace pour relever les défis liés à la distribution d'applications dans des environnements diversifiés.

**Implémentation :**

****

****

****

**CI/CD avec Jenkins**

pipeline {

agent any

tools {

maven 'maven'

}

stages {

stage('Git Clone') {

steps {

script {

checkout([$class: 'GitSCM', branches: [[name: 'master']], userRemoteConfigs: [[url: 'https://github.com/MohamedElam/microservices-consul.git']]])

}

}

}

stage('Build') {

steps {

script {

dir('salle-service') {

bat 'mvn clean'

bat 'mvn install -DskipTests'

}

dir('reservation-service') {

bat 'mvn clean'

bat 'mvn install -DskipTests'

}

dir('client-service') {

bat 'mvn clean'

bat 'mvn install -DskipTests'

}

dir('gateway-service') {

bat 'mvn clean'

bat 'mvn install -DskipTests'

}

}

}

}

stage('Create Docker Images') {

steps {

script {

dir('./salle-service') {

bat 'docker build -t salle-service .'

}

dir('./reservation-service') {

bat 'docker build -t reservation-service .'

}

dir('./client-service') {

bat 'docker build -t client-service .'

}

dir('./gateway-service') {

bat 'docker build -t gateway-service .'

}

}

}

}

stage('Run with Docker Compose') {

steps {

script {

bat 'docker-compose down || true'

bat 'docker-compose build || true'

bat 'docker-compose up -d'

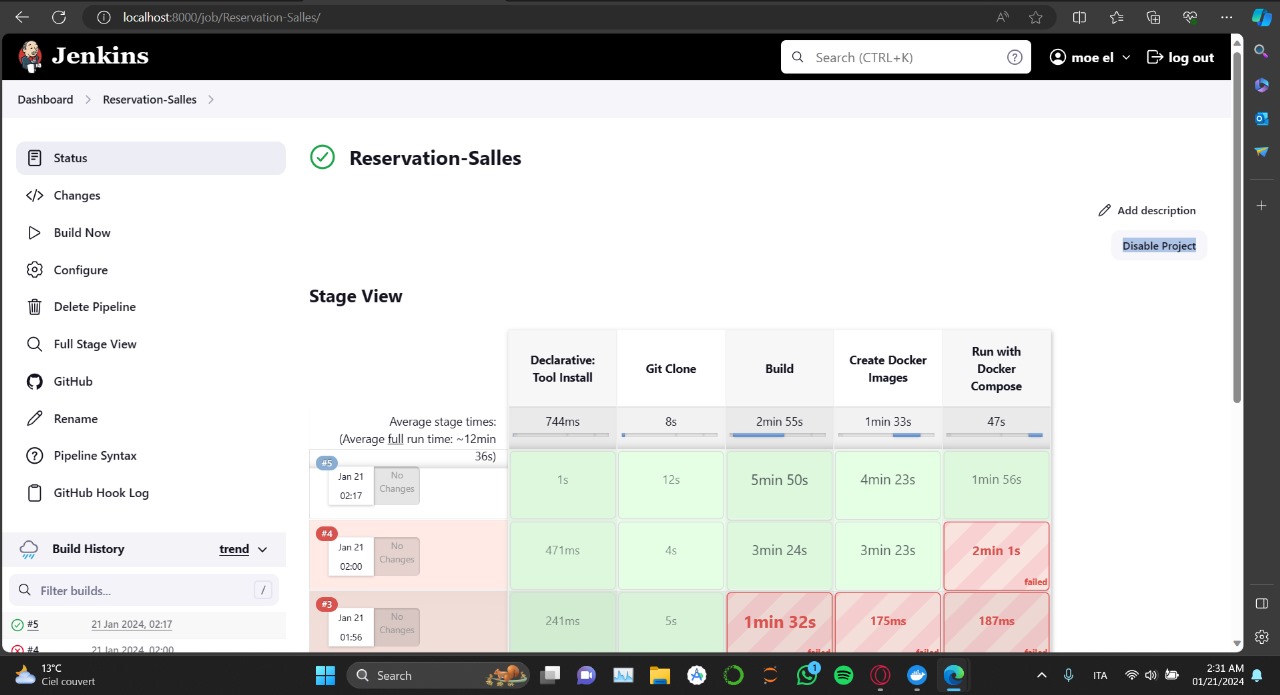
}

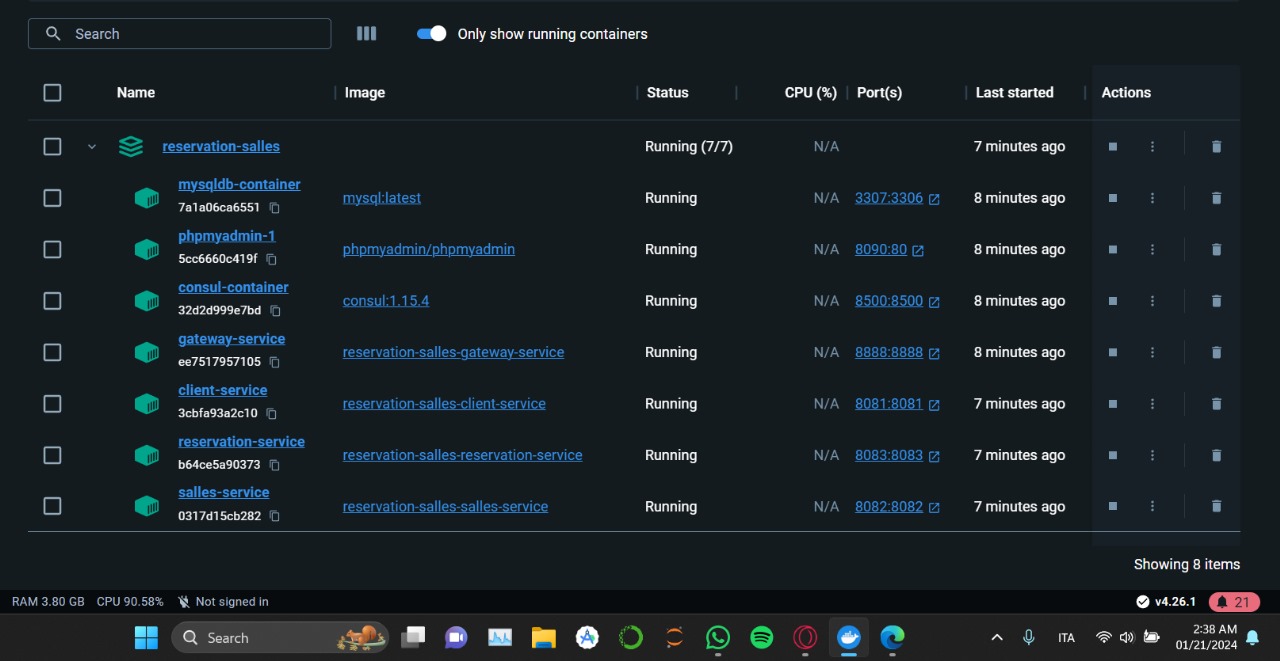
}

}

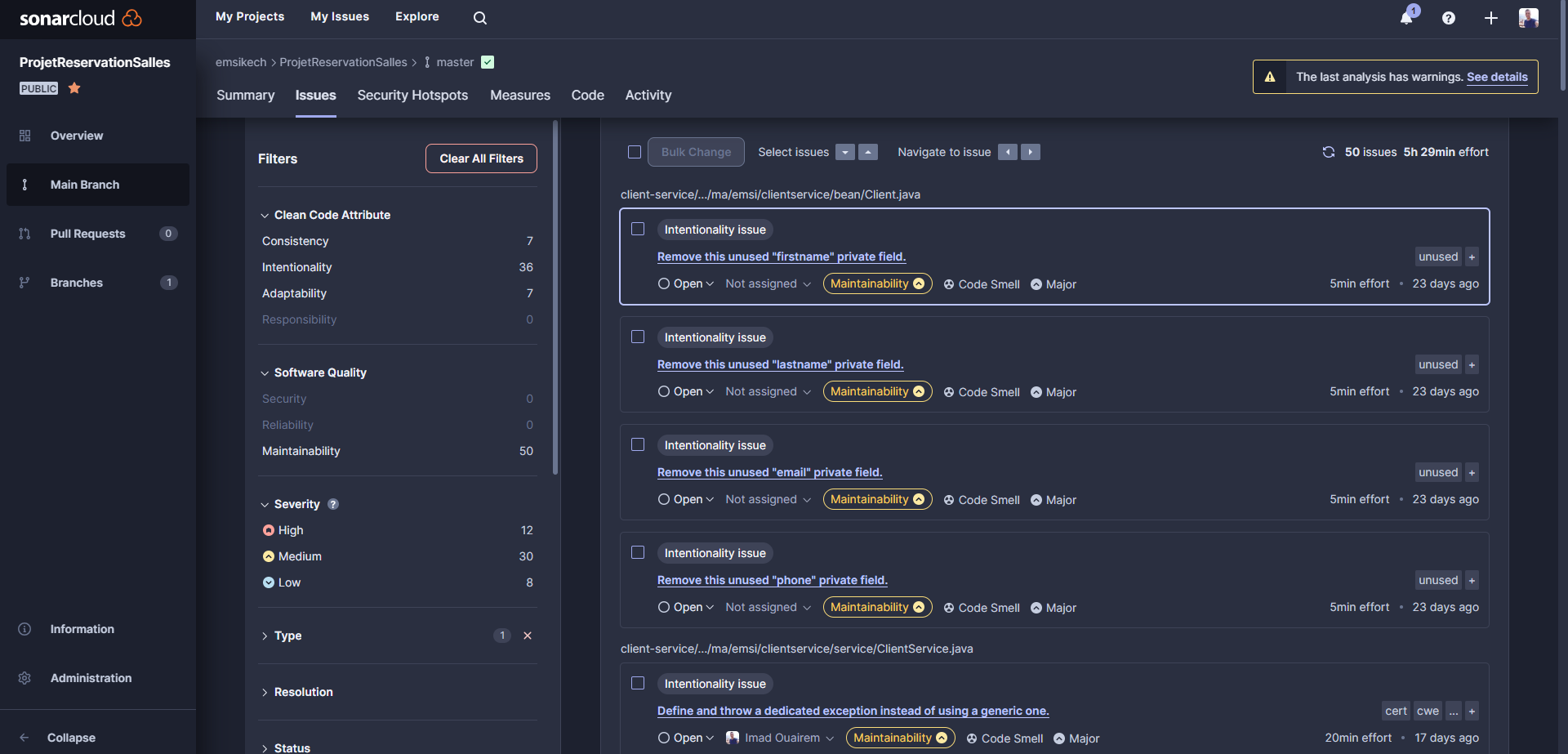
}

}





**Intégration de SonarQube**

****

**3) Implémentation**

**3.1 Méthodolige**

L'implémentation de notre application suit une méthodologie de développement en cascade, mettant l'accent sur une planification et une exécution linéaires des différentes phases du projet. Les étapes du développement, telles que la conception, la programmation, les tests, et le déploiement, sont abordées séquentiellement avec des jalons clairement définis avant de passer à la phase suivante. Cette approche offre une vision globale du projet dès le départ, permettant une gestion structurée et une allocation précise des ressources. En ce qui concerne la technologie utilisée, notre application repose principalement sur Java avec Spring Boot pour le back-end. Cette combinaison offre une architecture solide et évolutive, facilitant le développement d'applications robustes et performantes.

**3.1 Développement Back-End/Front-End**

**Développement Front-End :**

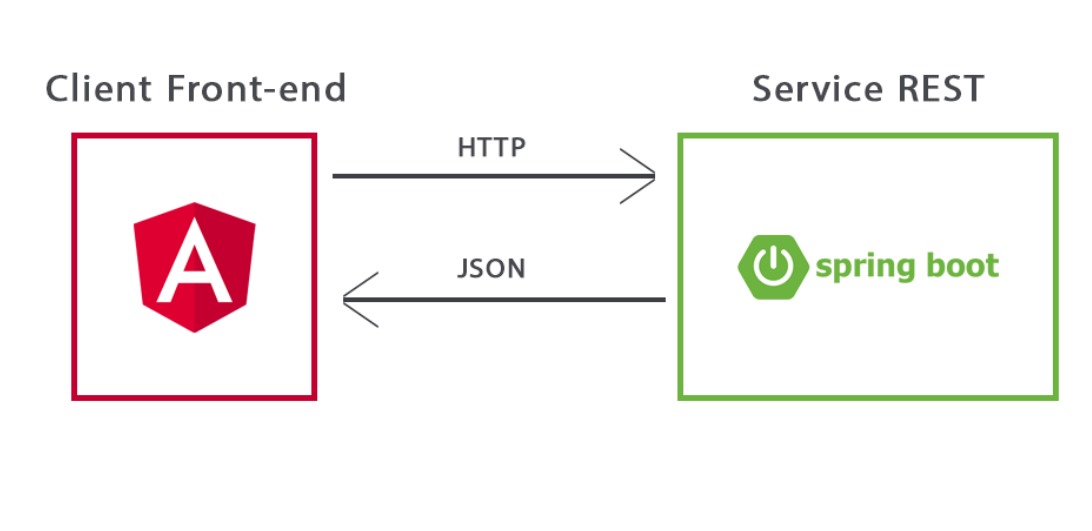
Pour le front-end de l'application, nous avons choisi d'utiliser Angular, un framework de développement puissant et polyvalent pour la création d'applications web dynamiques. Angular offre un ensemble riche de fonctionnalités qui nous permet de construire une interface utilisateur interactive et réactive, adaptée aux exigences modernes de navigation et d'expérience utilisateur.

La puissance d'Angular réside dans sa capacité à faciliter le développement de Single Page Applications (SPA), offrant ainsi une expérience utilisateur fluide et rapide, similaire à une application de bureau. Grâce à son architecture basée sur des composants, Angular nous permet de construire une interface structurée où chaque élément de l'application - comme les formulaires de réservation, les calendriers et les panneaux de gestion - est un composant indépendant, ce qui facilite la maintenance et les tests.

L'intégration avec les API REST du back-end est une autre force d'Angular, permettant une communication efficace et sécurisée avec les serveurs. Ceci est crucial pour des fonctionnalités telles que la récupération des disponibilités des salles, la gestion des réservations et l'affichage des notifications. De plus, Angular offre des outils avancés pour la gestion de l'état de l'application, le routage, et la validation des formulaires, ce qui contribue à une expérience utilisateur cohérente et sans faille.

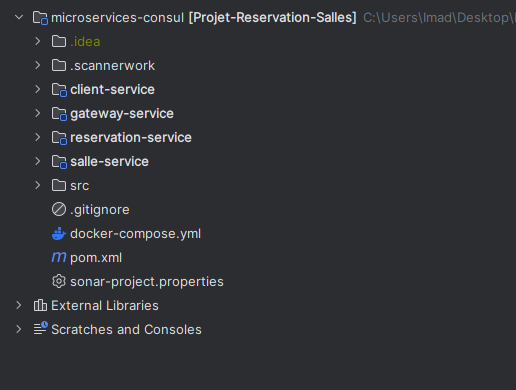
**Back-End :**

Le back-end est développé en Java en utilisant le framework Spring Boot, qui facilite la création d'applications robustes et évolutives. Spring Boot est utilisé pour gérer la logique métier, l'accès aux bases de données, et l'intégration des différents microservices. Docker est également utilisé pour conteneuriser les services, garantissant ainsi une implémentation et un déploiement cohérents et fiables.

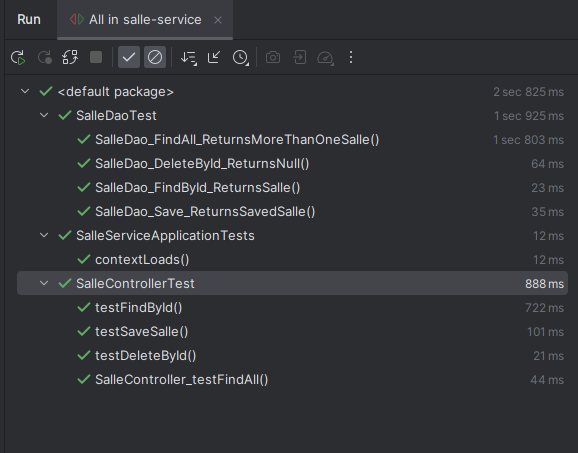


**Authentification et sécurité :**

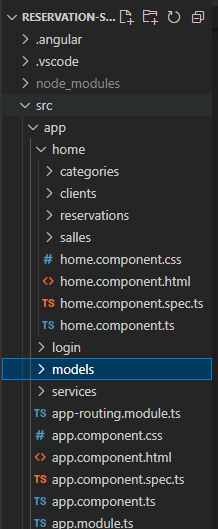
**Architecture du back-end :**

****

**Les tests unitaires :**

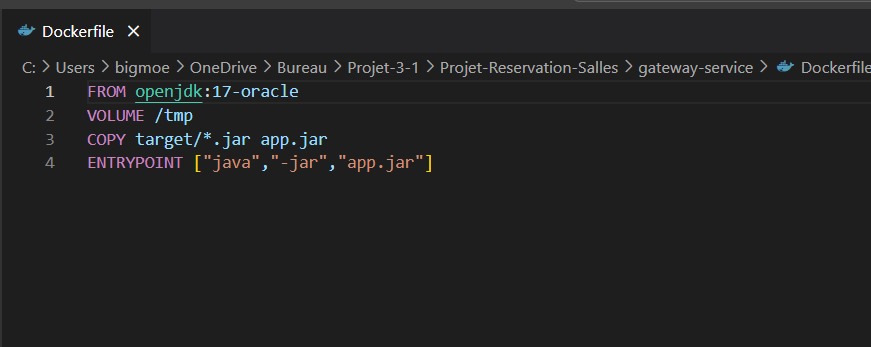
****

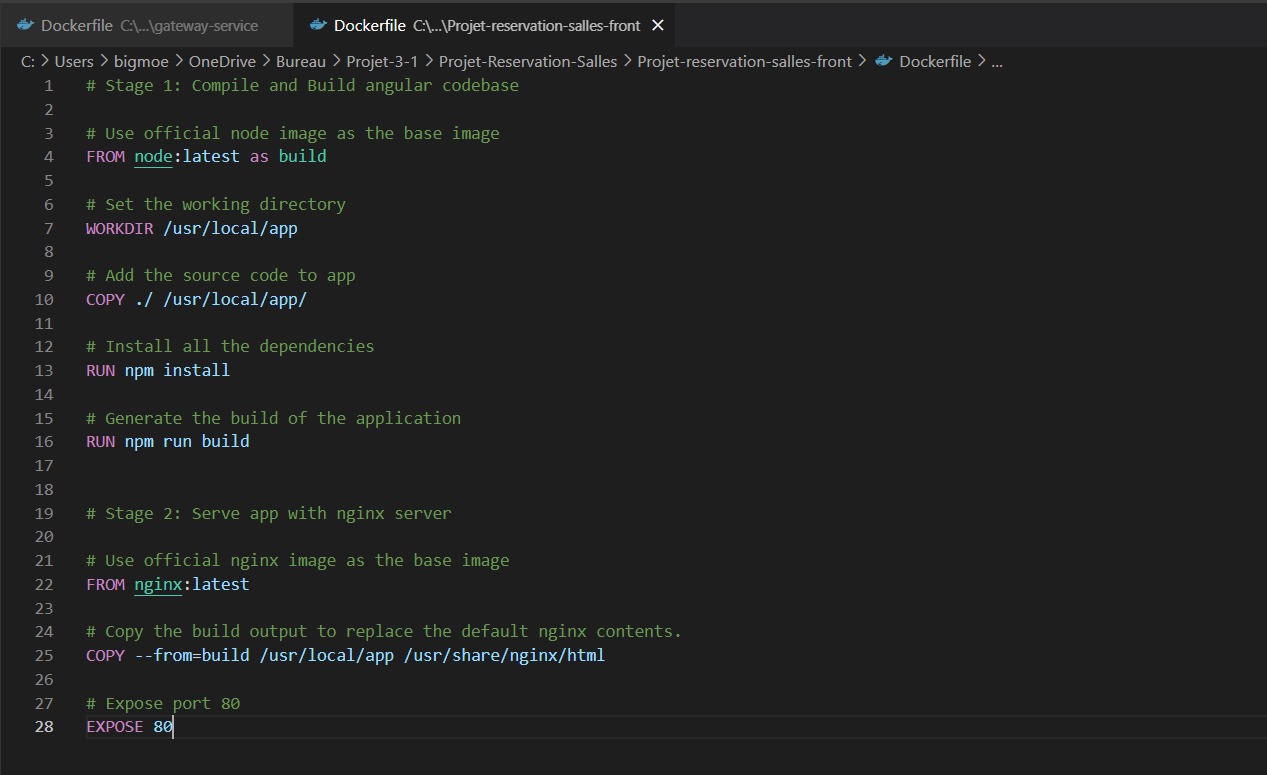
**Architecture du front-end :**

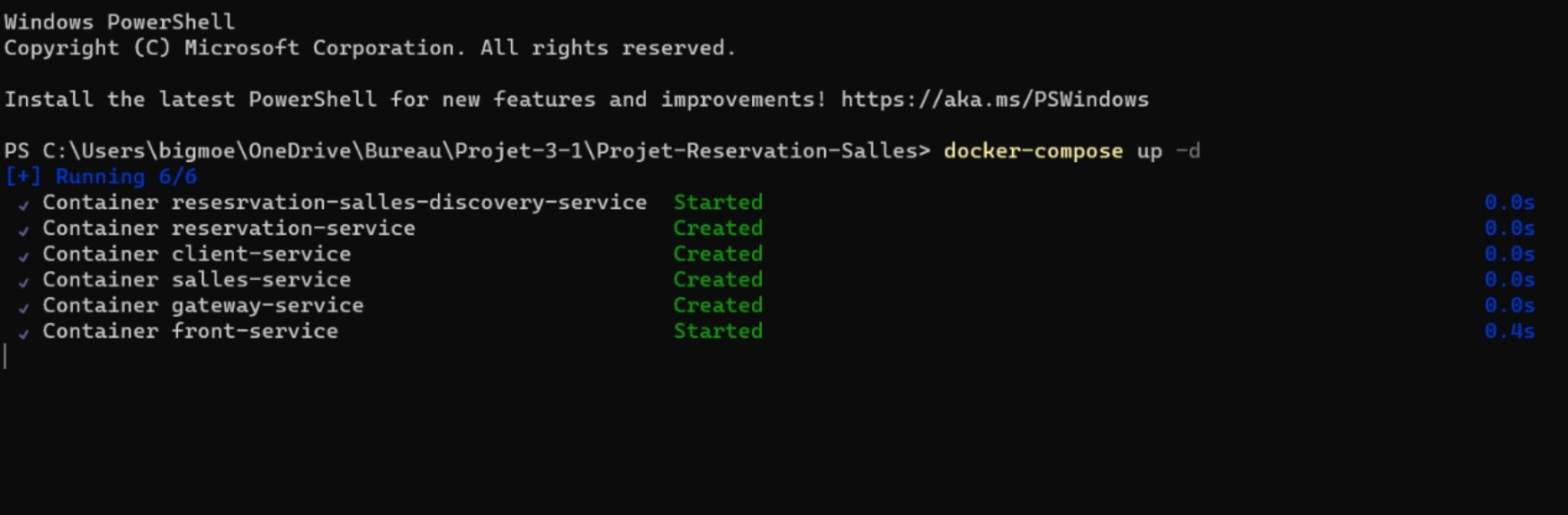


**Conteneurisation des service sur docker :**

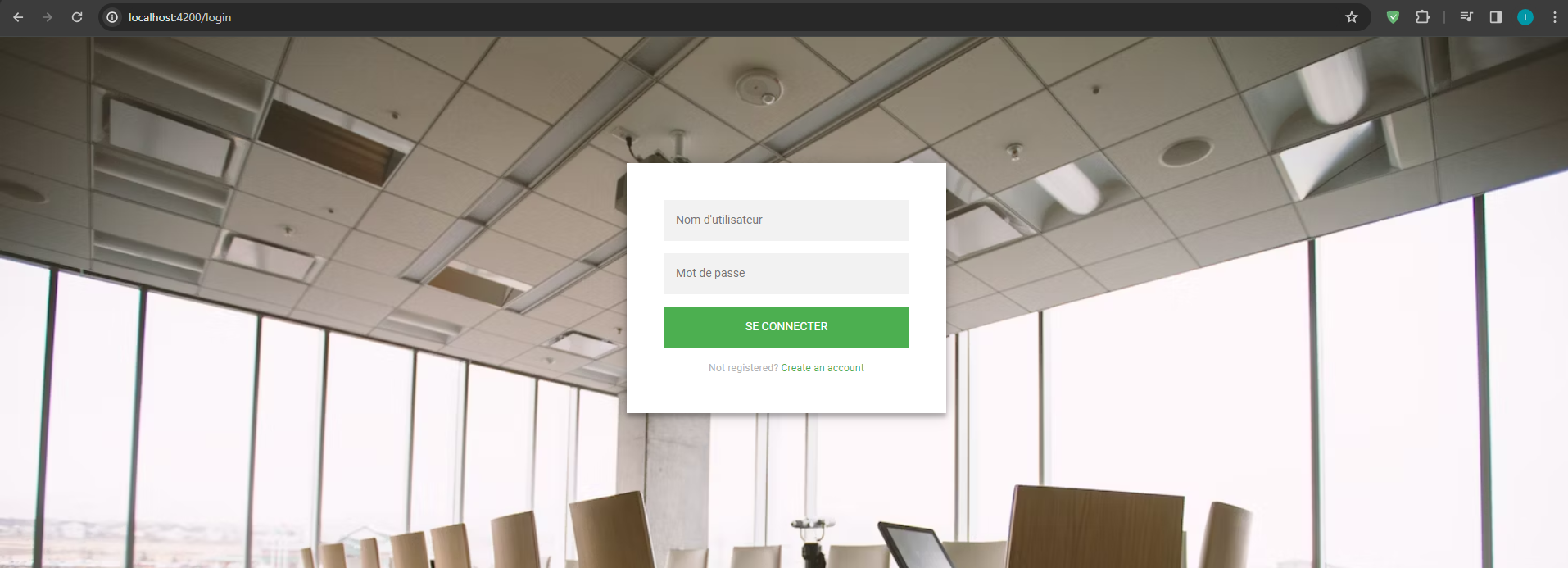
Ci-dessous, vous trouverez les captures d'écran de nos fichiers Docker, illustrant la configuration utilisée pour le déploiement de notre application front-end et back-end.

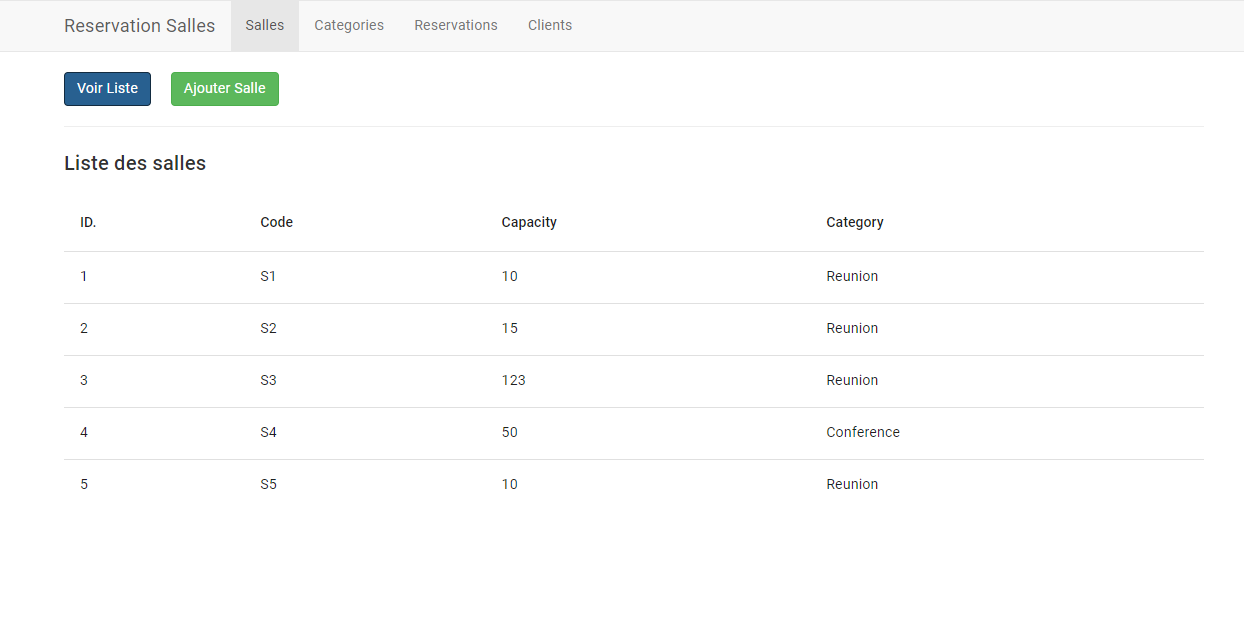


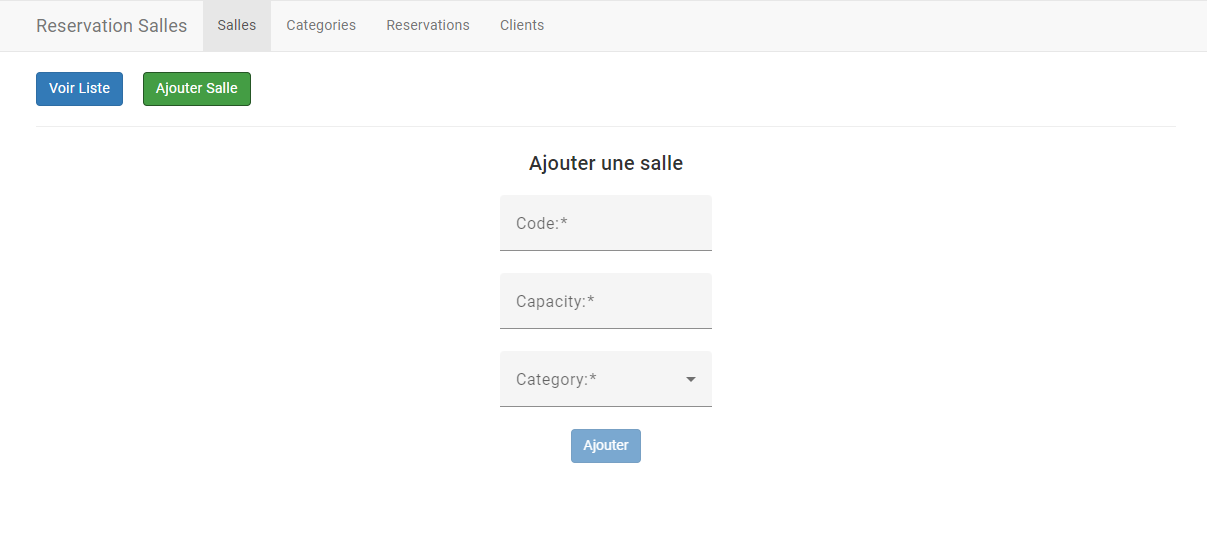


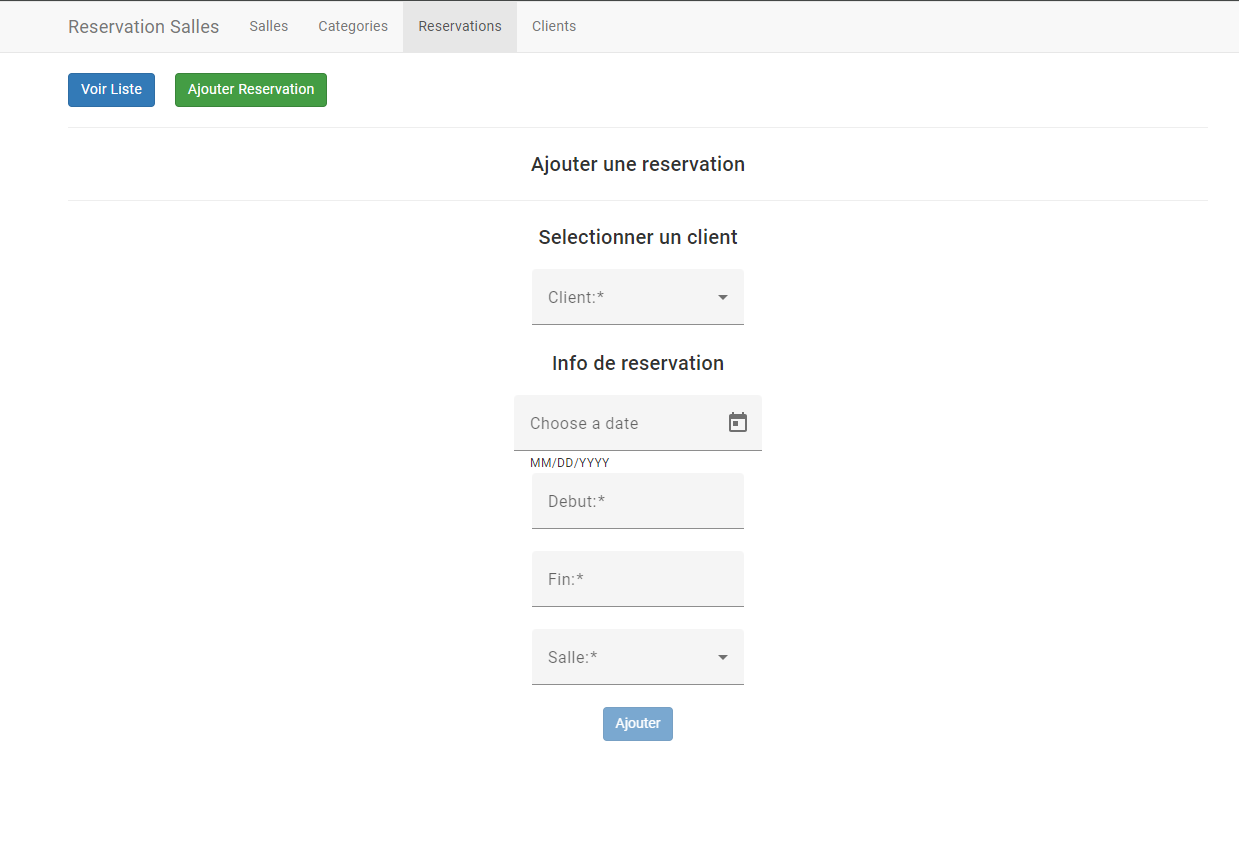


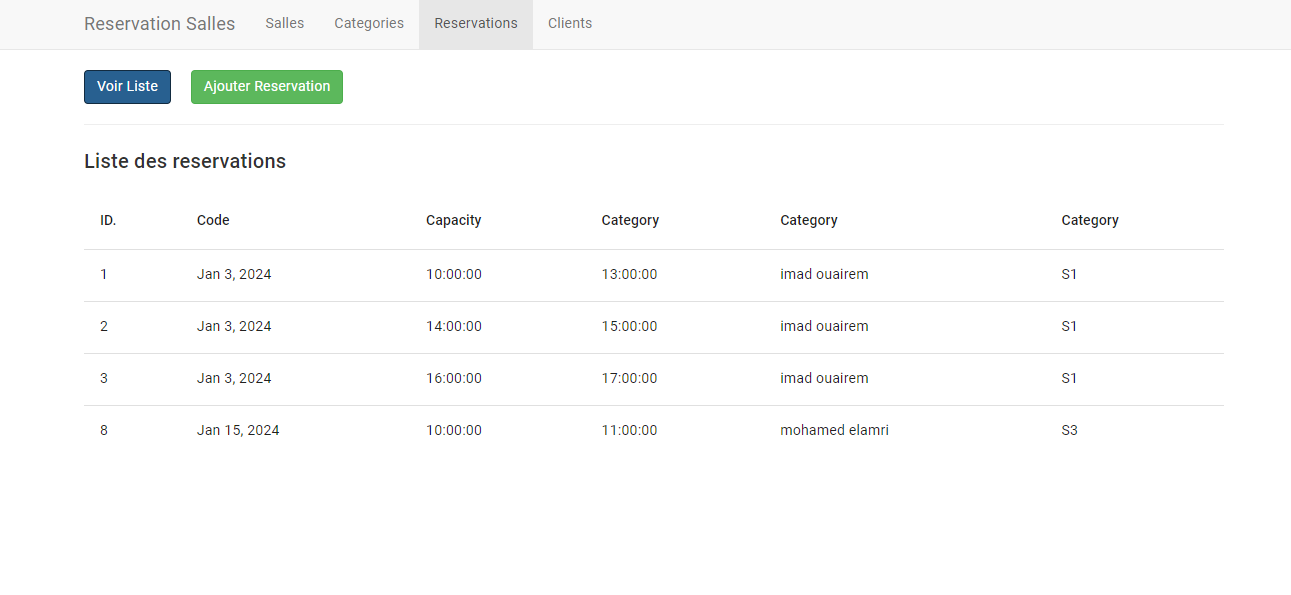
**Capture d'écran de l’application:**

****

****

****

****

****