

Module : Architecture des composants d'entreprise

Rapport Projet- Hotels Bookings

19 Janvier 2024 / 08:30 / SALLE 536

L'ÉQUIPE

Badreddine IBZAZNE, Oussama BOUHAMIDI, Ayoub CHAKIR

1. Introduction

Aperçu du projet: Le projet "**Hotels Booking**", a été initié dans le but de créer une application Web dédiée à la réservation d'hôtels. Dans un marché dynamique de l'hospitalité, il est essentiel de disposer d'une infrastructure logicielle flexible, évolutive et facilement maintenable pour répondre aux besoins des utilisateurs finaux. Le choix stratégique de cette application repose sur l'adoption d'une architecture microservices.









Importance de l'architecture microservices

L'architecture microservices a été sélectionnée en raison de ses nombreux avantages pour la conception, le développement et la maintenance des systèmes logiciels. Contrairement aux architectures monolithiques, les microservices permettent de découpler les différentes fonctionnalités de l'application en modules autonomes, favorisant ainsi la flexibilité et la scalabilité. Chaque service peut être développé, déployé et mis à l'échelle indépendamment, facilitant ainsi la gestion des mises à jour et l'ajout de nouvelles fonctionnalités sans perturber l'ensemble du système.

La modularité inhérente à l'architecture microservices permet également une meilleure répartition des responsabilités, favorisant le développement par équipes spécialisées et accélérant le cycle de développement. De plus, cette approche facilite la gestion des erreurs et la résilience, car une défaillance dans un service n'affecte pas nécessairement l'ensemble du système.

En résumé, l'architecture microservices offre une solution agile et adaptable, idéale pour un projet comme la "Hotels Booking" où la réactivité aux changements du marché et la facilité de maintenance sont des aspects cruciaux du succès.

Technologies utilisées

 mongoDB	 EXPRESS JS		
 Firebase Firestore	 docker		 sonarqube

Node.js : Un environnement d'exécution côté serveur basé sur le moteur JavaScript V8 de Google. Il permet d'exécuter du code JavaScript côté serveur, facilitant le développement d'applications côté serveur performantes et évolutives.

Express.js : Un framework web pour Node.js. Il simplifie la création d'API web et d'applications en fournissant des fonctionnalités et des abstractions utiles. Express.js est largement utilisé pour le développement rapide et la création de serveurs web robustes.

Angular : Un framework open-source développé par Google pour la création d'applications web dynamiques et interactives. Il simplifie la création de l'interface utilisateur et facilite la gestion de l'état de l'application grâce à des fonctionnalités telles que la liaison de données bidirectionnelle.

Firebase, Firestore : Firebase est une plateforme de développement d'applications mobiles et web proposée par Google. Il fournit divers services, y compris l'authentification, la base de données (Firestore), le stockage de fichiers, etc. Firestore est une base de données NoSQL en temps réel de Firebase, adaptée au stockage et à la récupération de données structurées.

MongoDB Atlas : MongoDB Atlas est un service de base de données cloud entièrement géré pour MongoDB. MongoDB est une base de données NoSQL qui stocke les données sous forme de documents JSON, offrant une flexibilité et une évolutivité considérables.

Docker : Docker est une plateforme de conteneurisation qui permet d'emballer une application et toutes ses dépendances dans un conteneur. Les conteneurs offrent une isolation légère, facilitent le déploiement et garantissent que l'application fonctionne de manière cohérente dans différents environnements.

Jenkins : Jenkins est un serveur d'intégration continue (CI) open-source. Il automatise le processus d'intégration et de

déploiement continu, permettant aux équipes de développement de détecter rapidement les erreurs et de livrer des logiciels de manière plus fiable.

SonarQube : SonarQube est une plateforme open-source dédiée à l'inspection continue de la qualité du code source. Il analyse le code pour détecter les violations des règles de qualité, les bugs potentiels, les vulnérabilités de sécurité, et fournit des rapports détaillés pour améliorer la qualité du code. SonarQube est souvent utilisé dans les pratiques de développement DevOps pour maintenir des normes de qualité élevées.

2. Architecture Microservices

Architecture

L'architecture de la "Hotels Booking" est basée sur le modèle microservices, où chaque composant fonctionne de manière autonome pour accomplir des tâches spécifiques, favorisant ainsi la modularité et la scalabilité du système.

Voici une vue d'ensemble des services qui composent l'application

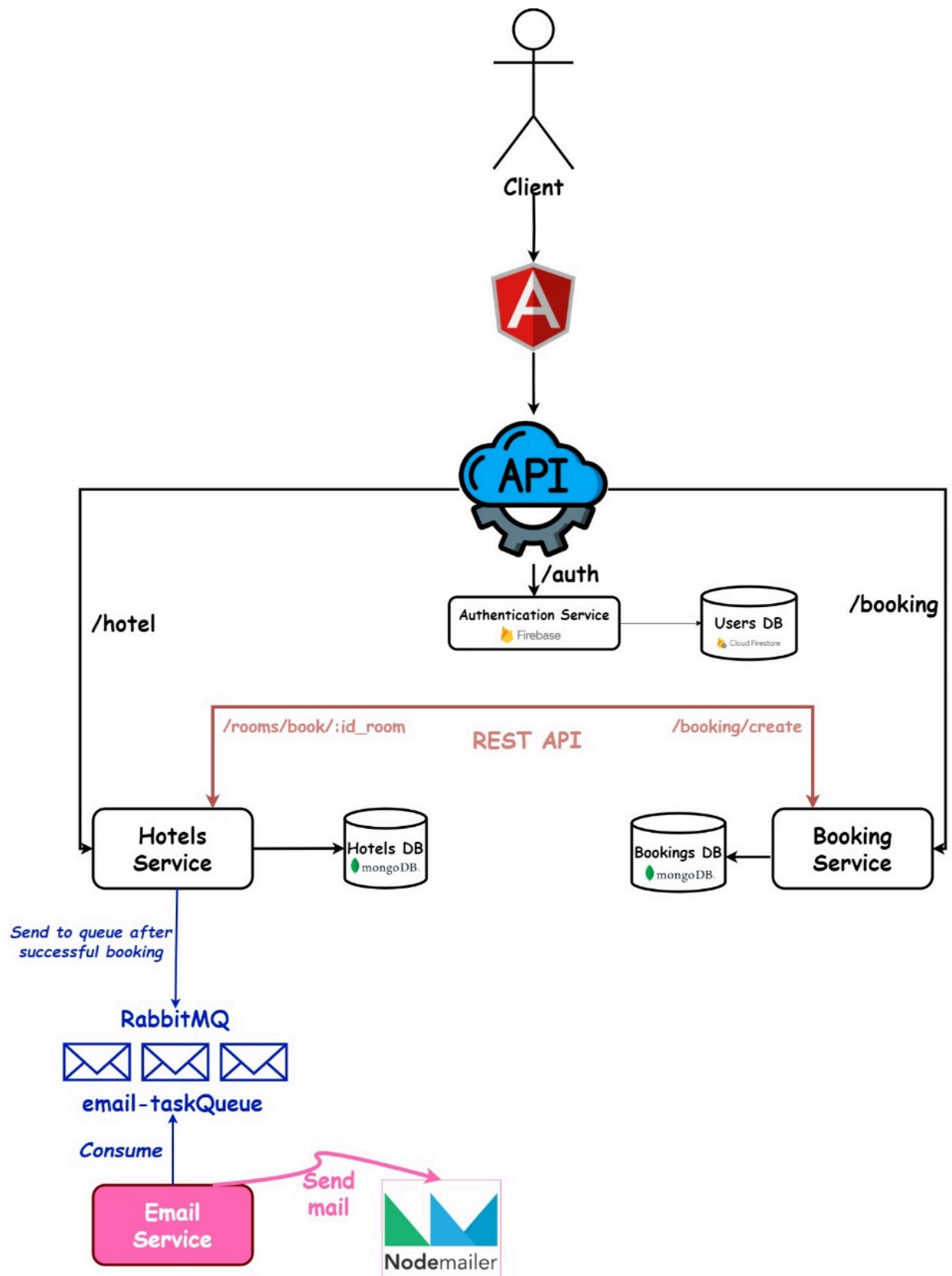


Figure 1 : Architecture Microservices

Description des services

★ Hotel Service

- ☑ Développé en utilisant Node.js et Express.js.
- ☑ Gère les fonctionnalités liées aux hôtels, offrant des opérations CRUD (Create, Read, Update, Delete) pour les hôtels.
- ☑ Interagit avec une base de données MongoDB pour stocker et récupérer les informations des hôtels.
- ☑ **La communication avec le Booking Service est sécurisée par la génération de tokens partagés entre les deux services.**

★ Auth Service : FIREBASE

- ☑ Utiliser Firebase pour l'authentification des utilisateurs.
- ☑ Fournit des fonctionnalités d'authentification et de gestion des utilisateurs.
- ☑ **Les APIs d'update et de delete sont protégées par une vérification de Firebase pour assurer la sécurité des opérations.**

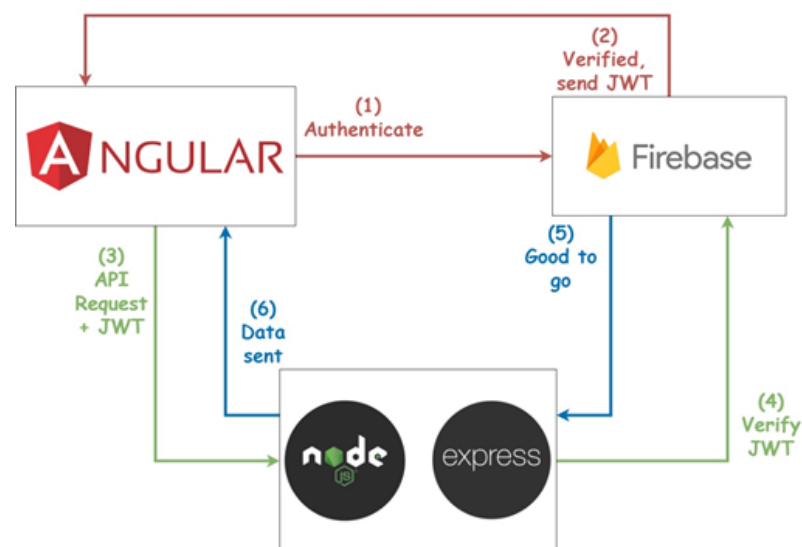


Figure 2 : Workflow Vérification de Firebase

★ Booking Service

- ☑ Développé en utilisant Node.js et Express.js.
- ☑ Gère les réservations d'hôtels en interagissant avec une base de données MongoDB.
- ☑ Fournit des API pour la gestion des réservations.

★ Email Service

- ☑ Notifie les clients par e-mail après une réservation.
- ☑ Fonctionne de manière asynchrone, intégrant **RabbitMQ** pour la communication entre services.

★ Frontend

- ☑ Développé en Angular.
- ☑ La communication avec les services est sécurisée par le processus d'authentification Firebase.
- ☑ **Les requêtes envoyées au backend incluent un token généré par Firebase pour chaque demande, permettant une vérification sécurisée du côté serveur.**

Mécanismes de communication

Les services de l'application interagissent de manière **synchrone** et **asynchrone**, en fonction des besoins spécifiques des fonctionnalités. Les mécanismes de communication comprennent :

→ Communication synchrone :

Utilisation d'**API REST** entre le service de réservation (Booking Service) et le service d'hôtel (Hotel Service), avec une sécurité renforcée par la génération et l'utilisation de tokens partagés.

→ Communication asynchrone :

Intégration de **RabbitMQ** pour la communication asynchrone entre le service d'hôtel (Hotel Service) et le service de

notification (Notification Service) lors de l'envoi d'e-mails de confirmation après une réservation.

Ces mesures de sécurité garantissent une architecture robuste et sécurisée pour la "Hotels Booking", avec une authentification en amont, des vérifications de token et des communications sécurisées entre services.

3. Conteneurisation avec Docker

Implémentation

La conteneurisation avec Docker constitue un élément central de l'architecture de la "Hotels Booking". Chaque service, qu'il s'agisse du service d'hôtel, du service de réservation, ou du service de notification, est encapsulé dans un conteneur Docker.

```
FROM node:14

# Set the working directory in the container
WORKDIR /usr/src/app

# Copy package.json and package-lock.json to the working directory
COPY package*.json ./

# Install application dependencies
RUN npm install

# Copy the application code to the container
COPY . .

# Expose the port on which the app will run
EXPOSE 5002

# Set environment variables
ENV NODE_ENV=production
ENV MONGO_BOOKING_SERVICE="mongodb+srv://hanslanda:halamadrid11@cluster0.zep2i0j.mongodb.net/booking-service?retryWrites=true&w=m"
ENV PORT=5002
ENV GOOGLE_APPLICATION_CREDENTIALS=./firebase/service-account.json
ENV HOTEL_API_SECRET="qBU4itbeb6Ri6AkKUwSWLy03sSMLeqMd"
ENV STRIPE_KEY="sk_test_51NzdLDz26lIMCWbTT8yCGN6THlxdeMFI1x00ALIAeypEJS15awJTxnv5a30Wztp2XQjkGFSM5jkr9c0IqbFq00p70pi65W"

# Command to run the application
CMD ["node", "index.js"]
```

Figure 3 : Exemple de fichier Dockerfile

Docker permet de créer, distribuer, et exécuter des applications dans des conteneurs légers, assurant la portabilité des environnements. Chaque conteneur inclut l'application, ses dépendances, et les bibliothèques

nécessaires, garantissant une exécution cohérente indépendamment de l'environnement d'hébergement.





























<input type="checkbox"/>	Name	Image	Status	CPU (%)	Port(s)	Last s	Actions
<input type="checkbox"/>	 hotel 52a310265055 	hanslanda1	Running	0%	5001:5001 	42 mir 	 
<input type="checkbox"/>	 notification 7b24d1016c6d 	hanslanda1	Running	0%	5005:5005 	53 mir 	 
<input type="checkbox"/>	 booking e3fdf78e2c32 	hanslanda1	Running	0.19%	5002:5002 	53 mir 	 
<input type="checkbox"/>	 gateway 2e9ad814d6c7 	hanslanda1	Running	0%	5555:5555 	17 mir 	 
<input type="checkbox"/>	 sonarqube-dockerized		Running (2/2)	0.7%		2 hour 	 

Figure 4 : Les conteneurs de microservices

Avantages

La conteneurisation avec Docker présente plusieurs avantages significatifs pour la "Hotels Booking" :

Portabilité : Les conteneurs Docker encapsulent l'application et ses dépendances, assurant une portabilité élevée entre les environnements de développement, de test, et de production.

Isolation : Chaque service fonctionne de manière isolée dans son propre conteneur, évitant les conflits de dépendances et assurant une stabilité accrue.

Efficacité du déploiement : La création rapide de conteneurs et l'utilisation de Docker Compose simplifient le déploiement de l'application dans divers environnements, accélérant ainsi le cycle de vie du développement.

Scalabilité : La nature légère des conteneurs permet une mise à l'échelle facile des services individuels en fonction des besoins de charge, assurant ainsi une gestion efficace des ressources.

Gestion des Versions : Docker facilite la gestion des versions des services, permettant un déploiement et une mise à jour cohérents de l'application sans perturber les autres composants.

En résumé, l'implémentation de la conteneurisation avec Docker dans la "Hotels Booking" offre une solution flexible, efficace, et évolutive, répondant aux exigences changeantes de l'environnement de développement et de déploiement.

4. CI/CD avec Jenkins

Processus et configuration

L'intégration continue (CI) et le déploiement continu (CD) sont des pratiques essentielles dans le cycle de développement de la "Hotels Booking". Jenkins, un serveur d'intégration continue, est utilisé pour automatiser ces processus, garantissant une livraison rapide et fiable des nouvelles fonctionnalités et mises à jour.

1. Déclenchement Automatique

Build Trigger : Le script utilise le déclencheur GitHub hook trigger for GITScm polling, ce qui signifie que le pipeline est automatiquement déclenché à chaque modification du référentiel GitHub.

2. Étape de Récupération des Sources Git

- ★ **git sources** : Cette étape récupère les sources du projet depuis le référentiel GitHub (<https://github.com/HansLanda14ib/hotels-booking-api.git>)

3. Analyse SonarQube

- ★ **SonarQube analysis** : Cette étape utilise SonarQube pour l'analyse statique du code source du service de réservation (booking). Le scanner SonarQube est configuré en utilisant l'outil défini dans l'environnement.

4. Vérification du Quality Gate de SonarQube

- ★ **Quality gate** : Cette étape attend la vérification du Quality Gate de SonarQube. Si la qualité du code ne répond pas aux critères définis, le pipeline sera interrompu.

5. Construction de l'Image Docker

- ★ **Build Docker Image** : Cette étape construit une image Docker à partir du code de chaque service.

6. Exécution du Conteneur Docker

- ★ **Run Docker** : Enfin, cette étape exécute un conteneur Docker nommé, en utilisant l'image Docker précédemment construite. Le conteneur est déployé en mode détaché (-d).

Avantages de CI/CD avec Jenkins

Rapidité : L'utilisation de Jenkins automatisé accélère le cycle de développement en permettant des déploiements rapides.

Fiabilité : Les vérifications de qualité avec SonarQube assurent une qualité constante du code source.

Visibilité : Les étapes du pipeline fournissent une visibilité claire sur le statut du processus CI/CD.

Scalabilité : Jenkins offre une solution scalable, permettant l'extension du processus CI/CD au fur et à mesure de l'évolution du projet.

Exemple de Script :

```

Script ?
1 pipeline {
2   agent any
3   tools {nodejs "NodeJs"}
4   stages {
5     stage('step1 > git sources') {
6       steps {
7         git 'https://github.com/HansLanda141b/hotels-booking-api.git'
8       }
9     }
10    stage('step2 > SonarQube analysis') {
11      environment {
12        SCANNER_HOME = tool 'SonarQubeScanner';
13      }
14      steps {
15        withSonarQubeEnv('SonarQube') {
16          dir('booking') {
17            bat "${SCANNER_HOME}/bin/sonar-scanner"
18          }
19        }
20      }
21    }
22    stage('step3 > Quality gate') {
23      steps {
24        script {
25          waitForQualityGate abortPipeline: true
26        }
27      }
28    }
29  }
30 }
31
32 stage('step4 > Build Docker Image') {
33   steps {
34     script {
35       dir('booking') {
36         bat 'docker build -t hanslanda14/bookingservice .'
37       }
38     }
39   }
40 }
41 stage('step5 > Run Docker') {
42   steps {
43     script {
44       bat "docker run --name booking -d -p 5002:5002 hanslanda14/bookingservice"
45     }
46   }
47 }
48 }
49
50

```

Figure 5 : Exemple de script de service Booking.

Ce script Jenkins garantit un processus de développement efficace, depuis l'intégration des changements jusqu'au déploiement en production, assurant ainsi une livraison continue et fiable de la "Hotels Booking".

Microservices Booking Hotels API

[Edit description](#)

All

+

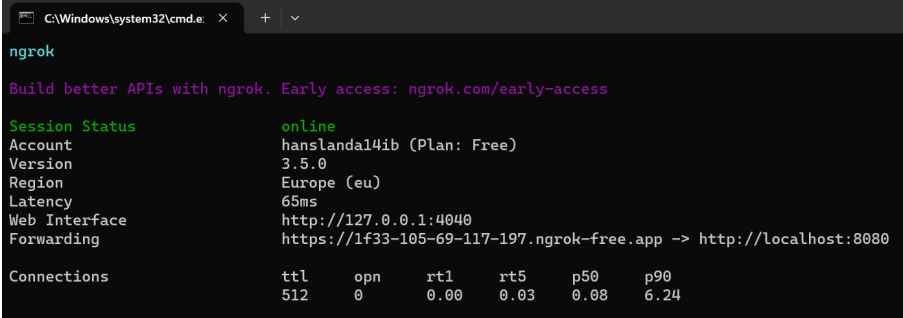
S	W	Name ↓	Last Success	Last Failure	Last Duration
✓	☁	bookings	2 hr 27 min #4	2 hr 33 min #1	1 min 7 sec ▶
✓	☁	gateway	54 min #7	59 min #5	1 min 22 sec ▶
✓	☀	hotel	1 hr 19 min #11	3 hr 7 min #6	42 sec ▶
✓	☀	notification	1 hr 30 min #1	N/A	1 min 16 sec ▶

Figure 6 : Dashboard Jenkins avec différents scripts.

5. Déploiement Automatique

Utilisation de Ngrok

Le déploiement automatique de la "Hotels Booking" s'effectue en utilisant l'outil **Ngrok**, qui permet de créer un tunnel sécurisé vers un serveur local, rendant ainsi l'application accessible depuis l'extérieur.



```
ngrok

Build better APIs with ngrok. Early access: ngrok.com/early-access

Session Status      online
Account             hanslanda14ib (Plan: Free)
Version             3.5.0
Region              Europe (eu)
Latency              65ms
Web Interface        http://127.0.0.1:4040
Forwarding            https://1f33-105-69-117-197.ngrok-free.app -> http://localhost:8080

Connections          ttl    opn    rt1    rt5    p50    p90
                    512    0      0.00   0.03   0.08   6.24
```

Figure 7 : Lancement Ngrok

Nous pouvons accéder à Jenkins en utilisant l'URL publique générée par Ngrok.

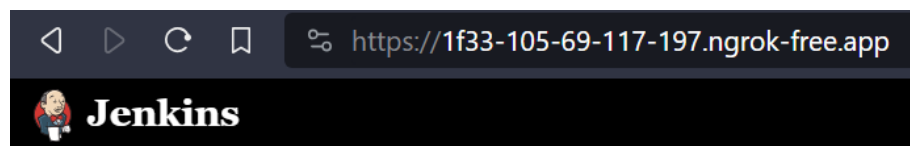


Figure 8 : Génération l'url jenkins publique

Avantages de l'Utilisation de Ngrok

- ★ **Simplicité** : Ngrok offre une solution simple pour exposer un service local sans avoir à configurer des infrastructures cloud complexes.
- ★ **Rapidité** : Le processus de déploiement est rapide, et l'application est immédiatement accessible via l'URL Ngrok.
- ★ **Accessibilité** pour le Développement : Ngrok facilite le partage de l'application en cours de développement avec des parties prenantes externes pour des tests ou des démonstrations.

- ★ **Flexibilité** : Ngrok peut être utilisé dans des environnements de développement, de test ou même pour des démonstrations temporaires.

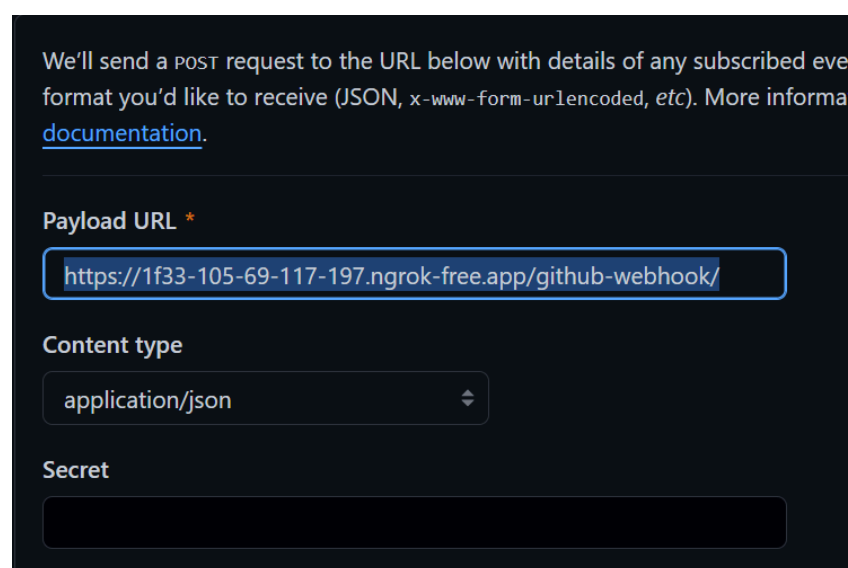
Configuration webhook

Un webhook est un moyen pour un dépôt GitHub de notifier une URL externe (dans ce cas, Jenkins) à chaque événement lié au dépôt.

Voici comment créer un webhook dans votre projet GitHub :

1. **Accédez à votre répertoire sur GitHub** : Rendez-vous sur la page principale de votre dépôt sur GitHub.
2. **Cliquez sur l'onglet "Settings" (Paramètres)** : Dans la barre de navigation de votre dépôt, cliquez sur l'onglet "Settings".
3. **Sélectionnez "Webhooks"** : Sur la page des paramètres, choisissez l'option "Webhooks" dans la colonne de gauche.
4. **Cliquez sur "Add webhook" (Ajouter un webhook)** : Généralement situé en haut de la page des webhooks.
5. **Configurez le webhook** :

Payload URL : Entrez l'URL de votre Jenkins.



We'll send a POST request to the URL below with details of any subscribed event in the format you'd like to receive (JSON, x-www-form-urlencoded, etc). More information in the [documentation](#).

Payload URL *

Content type

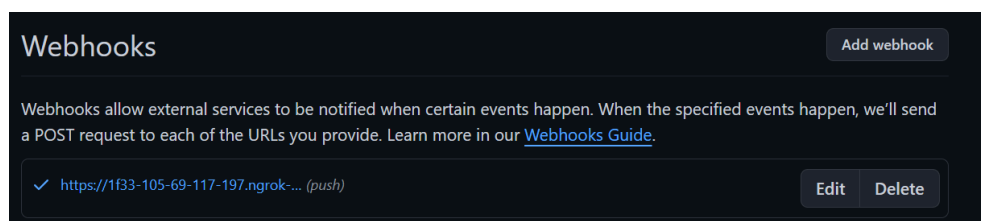
Secret

Figure 9 : Configuration Webhook GitHub

Content type : Choisissez application/json.

Which events would you like to trigger this webhook? :
Choisissez l'événement qui déclenche votre pipeline (par exemple, "Just the push event" pour déclencher lors d'un push).

6. **Cliquez sur "Add webhook" (Ajouter un webhook) :** Une fois que vous avez configuré les paramètres, cliquez sur le bouton pour ajouter le webhook.



Dans Jenkins manage/configure, ajouter l'url github:

GitHub Pull Requests

Published Jenkins URL

☒ Actualise local repo on factory creation

Figure 10 : Ajouter lien du projet en Jenkins

6. Intégration de SonarQube

Configuration

L'intégration de SonarQube dans le processus CI/CD de la "Hotels Booking" vise à améliorer la qualité du code en identifiant les problèmes potentiels et en fournissant des analyses statiques approfondies. Voici comment la configuration de SonarQube est gérée dans le pipeline Jenkins

1. Création du Projet dans SonarQube

- Accédez à votre instance de SonarQube.
- Connectez-vous à votre compte SonarQube.
- Créez un nouveau projet pour votre application.

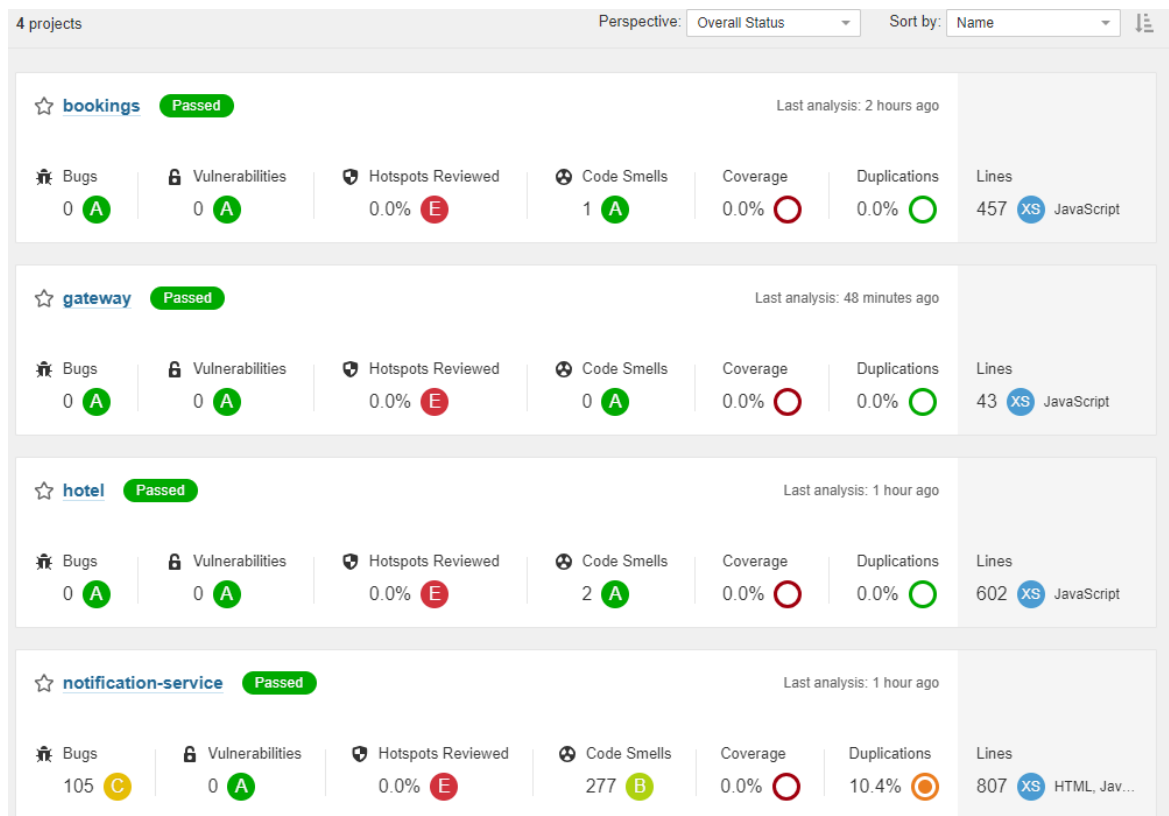


Figure 10 : Analyse des projets par sonarqube

2. Génération du Token d'Accès

- Allez dans les paramètres utilisateur dans SonarQube.
- Accédez à l'onglet "Security".
- Générez un token d'accès pour votre projet.

Name
SonarQube

Server URL
Default is http://localhost:9000
http://localhost:9000

Server authentication token
SonarQube authentication token. Mandatory when anonym
lastone

+ Add ▾

Advanced ▾

3. Configuration du Projet dans le Fichier `sonar-project.properties`

Créez un fichier nommé `sonar-project.properties` à la racine de votre projet avec les informations nécessaires :

```
1 sonar.projectKey=bookings
2 sonar.sources=src
3 sonar.login=2440e86ae90739ecf967711437f701c5ebba3f4c
4 sonar.sourceEncoding=UTF-8
5 sonar.host.url=http://localhost:9000
6 sonar.exclusions=./node_modules/*,./firebase/*,src/configs/*
```

Figure 11 : Exemple de fichier de config pour une app Node.JS

Avec ces étapes, chaque fois que votre pipeline Jenkins sera déclenché, il effectuera automatiquement l'analyse SonarQube avec les configurations définies dans le fichier `sonar-project.properties`.

Cette approche assure une intégration transparente de l'analyse de qualité du code dans votre processus CI/CD.

Bénéfices pour la qualité du code

Détection des Problèmes de Code : SonarQube identifie les problèmes potentiels tels que les bugs, les vulnérabilités de sécurité, les erreurs de codage, et fournit des suggestions pour les améliorations.

Analyse Statique Approfondie : L'outil effectue une analyse statique approfondie du code source, évaluant la qualité du code selon des normes prédéfinies.

Intégration Transparente : Intégré dans le pipeline CI/CD, SonarQube offre une intégration transparente, fournissant des résultats immédiats après chaque commit.

Historique de Qualité du Code : SonarQube conserve un historique de la qualité du code, permettant de suivre les améliorations ou les dégradations au fil du temps.

8. Conclusion

Résumé des accomplissements

La mise en œuvre de l'architecture microservices dans la "Hotels Booking" a permis d'accomplir plusieurs objectifs majeurs. Les principales réalisations comprennent :

Microservices Architecture : Adoption d'une architecture basée sur des microservices pour favoriser la modularité, la scalabilité et la maintenance aisée des différents services.

Services Développés :

- ☑ **Hotel Service** : Gestion des fonctionnalités liées aux hôtels, construit avec Node.js, Express.js, et MongoDB.
- ☑ **Auth Service** : Gestion de l'authentification avec Firebase et Firestore.
- ☑ **Booking Service** : Gestion des réservations, développé avec Node.js, Express.js, et MongoDB.
- ☑ **Notification Service** : Service de notification par e-mail pour informer les utilisateurs après une réservation.
- ☑ **Frontend** : Interface utilisateur conviviale développée en Angular.

Conteneurisation avec Docker : Utilisation de Docker pour encapsuler chaque service, facilitant le déploiement, la gestion des dépendances et la portabilité.

CI/CD avec Jenkins : Automatisation du processus d'intégration continue et de déploiement continu, assurant une livraison rapide et fiable.

Déploiement Automatique avec Ngrok : Exposition automatique de l'application via Ngrok pour faciliter l'accès lors des phases de développement et de test.

Intégration de SonarQube : Intégration d'analyses statiques approfondies avec SonarQube pour garantir une qualité de code élevée.

Perspectives futures

Les perspectives futures pour la "Hotels Booking" incluent plusieurs axes d'amélioration et de développement continu :

Tests Automatisés : Développement de suites de tests automatisés approfondis pour garantir la robustesse et la stabilité des services.

Migration vers des Services Cloud Gérés : Évaluation et éventuelle migration vers des services cloud gérés comme Azure Cosmos DB, Firebase Realtime Database, ou d'autres solutions en fonction des besoins évolutifs.

Optimisation des Performances : Optimisation continue des performances des microservices pour garantir une expérience utilisateur rapide et fluide.

En conclusion, les réalisations actuelles posent les bases d'une application robuste et évolutive. Les perspectives futures visent à renforcer ces fondations et à répondre aux exigences changeantes du marché et des utilisateurs.