

# ENERDRIVE

APPLICATION MICROSERVICES AVEC SPRINGBOOT  
REACTJS A L'AIDE DE JUNIT SNOARQUBE ET INTEGRATION  
CONTINUE JENKINS



ARCHITECTURE MICROSERVICE RAPPORT | 20 JAN 2024  
PRESENTE PAR HOUSSAM NAASSE ET HOUSSAM HARAF  
ENCADRE PAR MR LACHGAR MOHAMED

# INTRODUCTION

EnerDrive est une application web qui vise à calculer la consommation d'énergie de chaque voiture d'un utilisateur, tout en fournissant une vue et statistique globale.

Les microservices sont une approche architecturale de développement logiciel dans laquelle une application est construite comme un ensemble de services indépendants et interopérables. Chaque service, également appelé microservice, est une unité autonome qui effectue une fonction spécifique de l'application.

# JUNIT

JUnit est un framework de test unitaire pour le langage de programmation Java. Il fournit des annotations pour identifier les méthodes de test, des assertions pour vérifier les résultats attendus, et un ensemble de règles pour personnaliser le comportement des tests. JUnit est largement utilisé dans le développement logiciel Java pour automatiser et simplifier les tests unitaires, ce qui permet aux développeurs d'identifier rapidement les erreurs dans leur code et de s'assurer que les différentes parties d'un programme fonctionnent correctement.



## Mockito

Mockito est une bibliothèque Java utilisée pour créer des objets simulés (mocks) dans le cadre des tests unitaires. Ces mocks sont utilisés pour remplacer des parties spécifiques d'un système afin d'isoler le code que vous testez. Mockito facilite la création de simulations d'objets et la définition de leur comportement lors des tests. Il permet aux développeurs de créer des tests plus fiables et plus prévisibles en éliminant les dépendances externes, en se concentrant sur l'unité de code testée, et en permettant la vérification du comportement attendu des objets simulés.



# SELENIUM

Selenium est un ensemble d'outils open source utilisés pour automatiser les tests fonctionnels des applications web. Il prend en charge plusieurs langages de programmation, notamment Java, C#, Python, Ruby, et JavaScript, ce qui permet aux développeurs de choisir le langage qui leur convient le mieux.

Selenium est principalement utilisé pour tester les fonctionnalités d'une application web en simulant le comportement d'un utilisateur réel.



# SONARQUBE

SonarQube est une plateforme open source destinée à l'analyse continue de la qualité du code source. Elle fournit des outils complets pour mesurer et améliorer la qualité du code, identifier les problèmes potentiels, et suivre l'évolution de la qualité du logiciel au fil du temps.

SonarQube prend en charge divers langages de programmation tels que Java, C#, JavaScript, Python, et d'autres, ce qui en fait un outil polyvalent pour les équipes de développement travaillant sur des projets variés.



## SPRINGBOOT

Spring Boot est un module open-source pour le développement rapide d'applications Java basées sur le framework Spring. Il vise à simplifier le processus de configuration et de déploiement des applications en offrant des conventions de développement et des outils intégrés.



## REACTJS

React est une bibliothèque JavaScript open-source développée par Facebook, utilisée pour la création d'interfaces utilisateur interactives et dynamiques. Il permet de construire des applications web modernes en utilisant une approche basée sur des composants réutilisables.



## JENKINS

Jenkins est un serveur d'intégration continue open-source largement utilisé dans le développement logiciel. Il automatise le processus d'intégration, de test et de déploiement des applications en continu, ce qui permet aux équipes de développement de détecter rapidement les erreurs et de livrer des logiciels de manière efficace et fiable.



# Jenkins

## NGROK

Ngrok est un outil open-source qui permet de créer un tunnel sécurisé entre un serveur local et Internet, en exposant localement un service ou une application à l'extérieur de votre réseau local.

# ngrok

# LES TEST UTINTAIRES

## TEST POUR LE SERVICE CAR

Cette classe de test assure que les méthodes de la classe CarService interagissent correctement avec son repository (CarRepository) en utilisant des mocks, et que les résultats retournés sont conformes aux attentes.

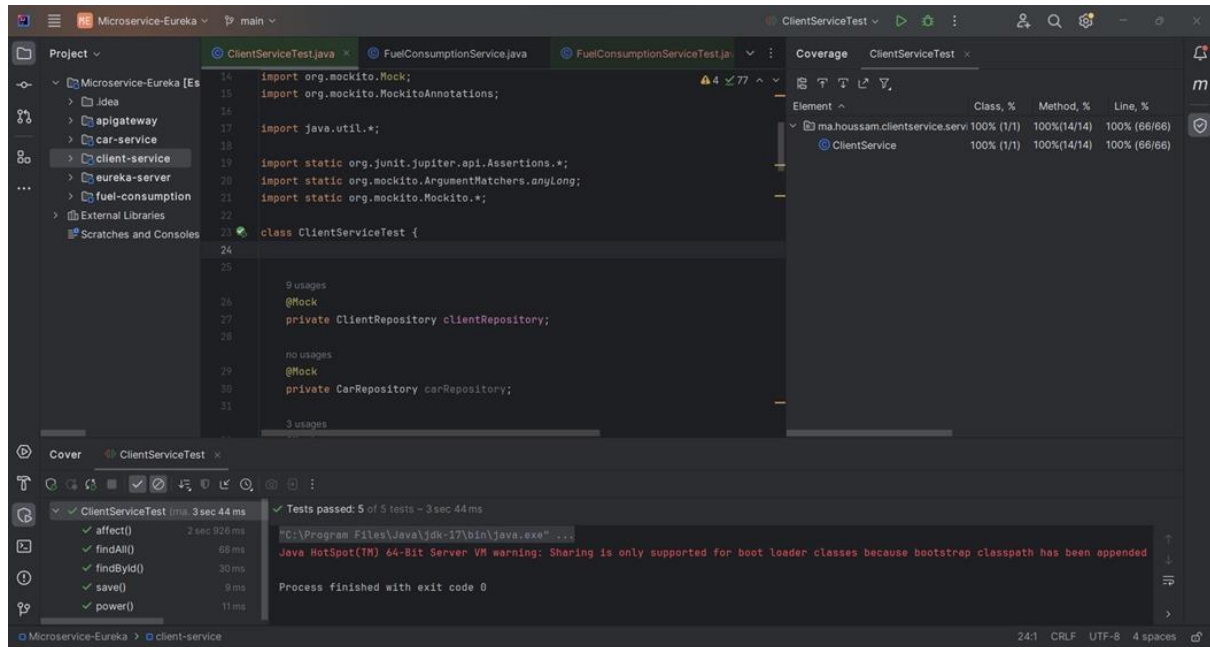
The screenshot shows an IDE with the following components:

- Project Explorer:** Shows the project structure with folders like `apigateway`, `car-service`, `src`, `main`, `java`, `ma.houssam.carservice`, `controller`, `entities`, `repository`, `service`, `resources`, `test`, and `target`.
- Code Editor:** Displays the `CarServiceTest.java` file. The code includes imports for `java.util.Arrays`, `java.util.List`, `java.util.Optional`, and `org.junit.jupiter.api.Assertions.*`. It defines a `CarServiceTest` class with a `@Mock` `CarRepository`, a `@InjectMocks` `CarService`, and a `@BeforeEach` `setUp()` method that calls `MockitoAnnotations.initMocks(this)`. The `@Test` method is also present.
- Coverage:** Shows a table with the following data:

Element	Class, %	Method, %	Line, %
ma.houssam.carservice.service	100% (1/1)	100% (3/3)	100% (3/3)
- Test Results:** Shows the results of the `CarServiceTest` run. The tests passed are `car()`, `cars()`, and `save()`. The total time taken for the tests is 2 sec 725 ms.

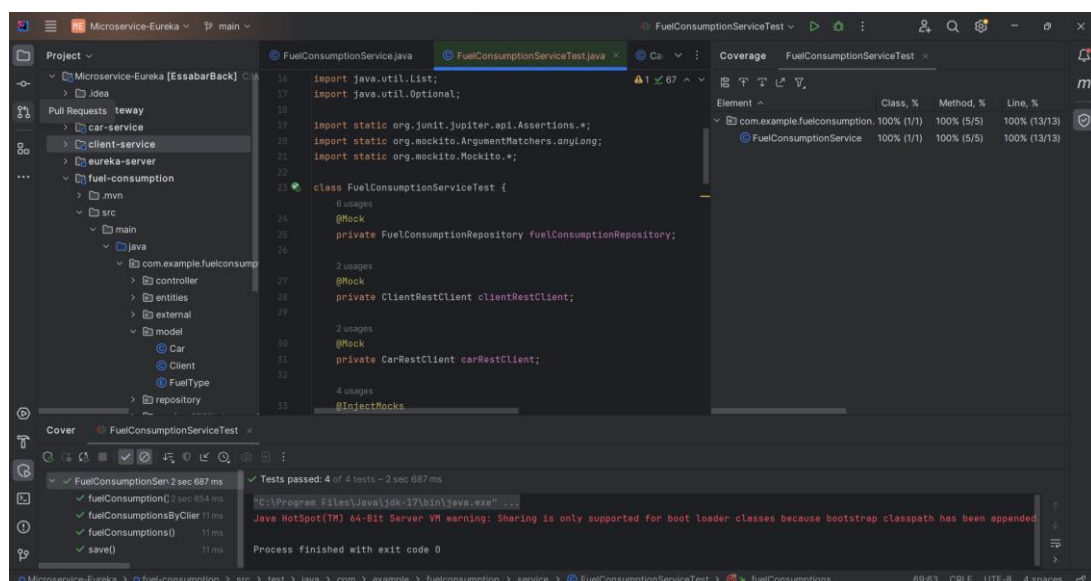
## TEST POUR LE SERVICE CLIENT

Chaque méthode de test suit une structure similaire, préparant les données nécessaires, définissant le comportement des mocks, appelant la méthode à tester, vérifiant que les mocks ont été utilisés correctement, puis effectuant des assertions sur le résultat attendu.



## TEST POUR LE SERVICE CONSOMMATION

Cette classe de test assure que les fonctions de la classe `FuelConsumptionService` interagissent de manière appropriée avec ses dépendances, à savoir les repositories et les clients externes, en utilisant des simulations (mocks). Elle vérifie également que les résultats renvoyés sont conformes aux résultats attendus.

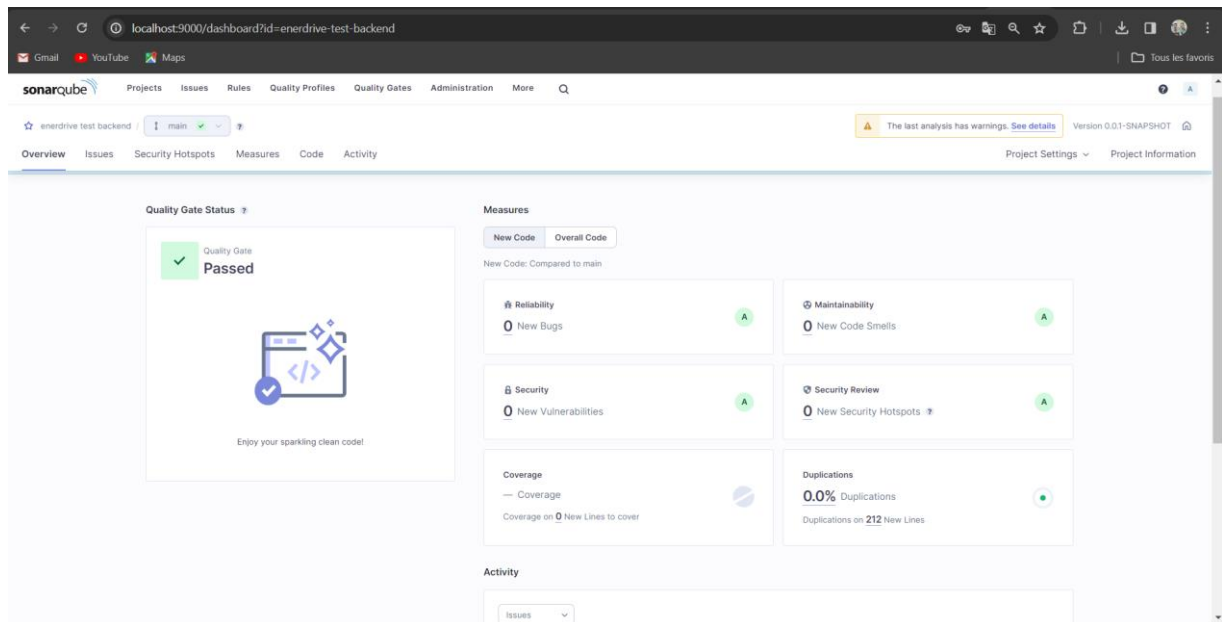




# ANALYSE DE CODE AVEC SONARQUBE

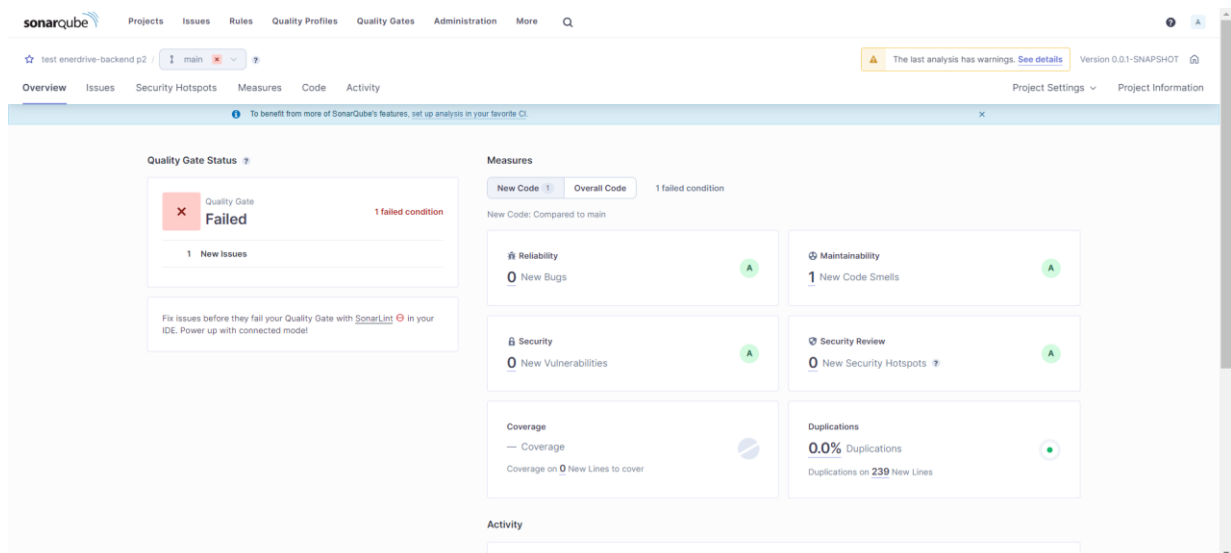
## TEST POUR CAR SERVICE

Le test de service pour la classe Car est une étape essentielle dans le processus de développement logiciel visant à garantir la qualité du code. Lorsqu'évalué par SonarQube, cet ensemble de tests est analysé pour détecter d'éventuelles anomalies, violations de bonnes pratiques de codage, et autres aspects liés à la qualité du code. Le test est bien passé.



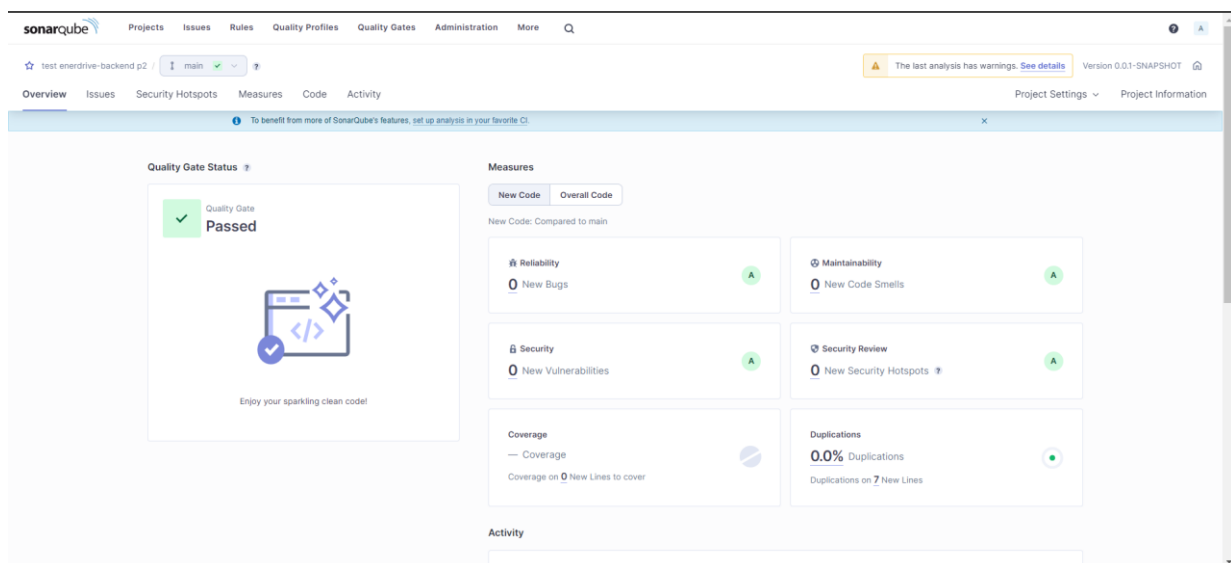
## TEST POUR CLIENT SERVICE

Le test de service pour la classe ClientService représente une étape cruciale dans la stratégie de garantie de qualité du code. Lors de son évaluation par SonarQube, cet ensemble de tests est minutieusement analysé pour déceler d'éventuelles irrégularités, violations de bonnes pratiques de codage et autres aspects liés à la qualité du code. Ici on a des erreurs au niveau de la maintenabilité



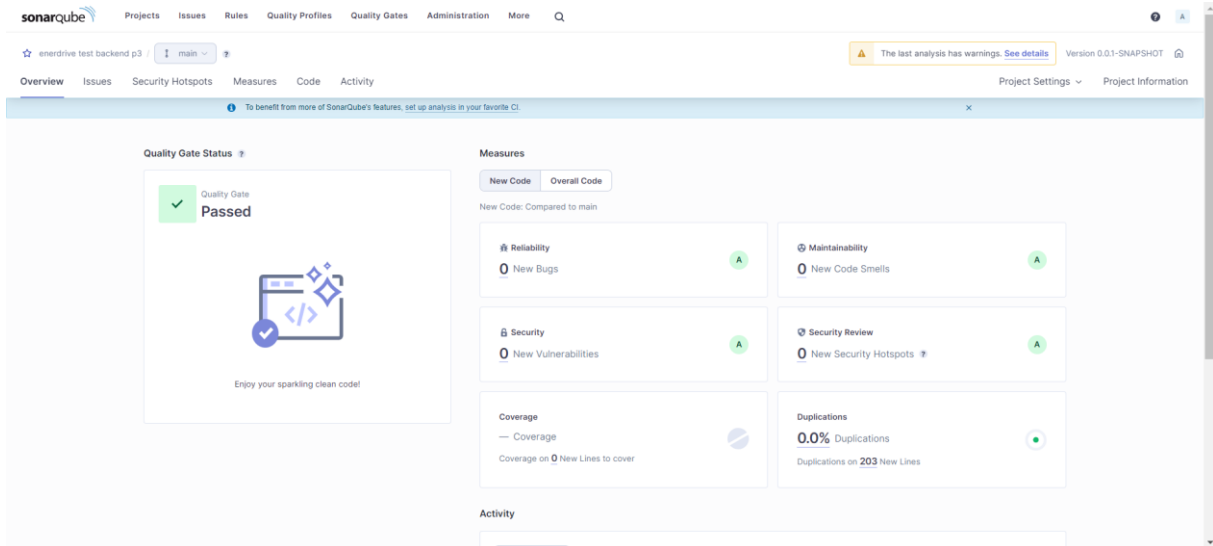
## APRES LA CORRECTIONS DES ERREURS

Après la correction des erreurs le test a bien passé



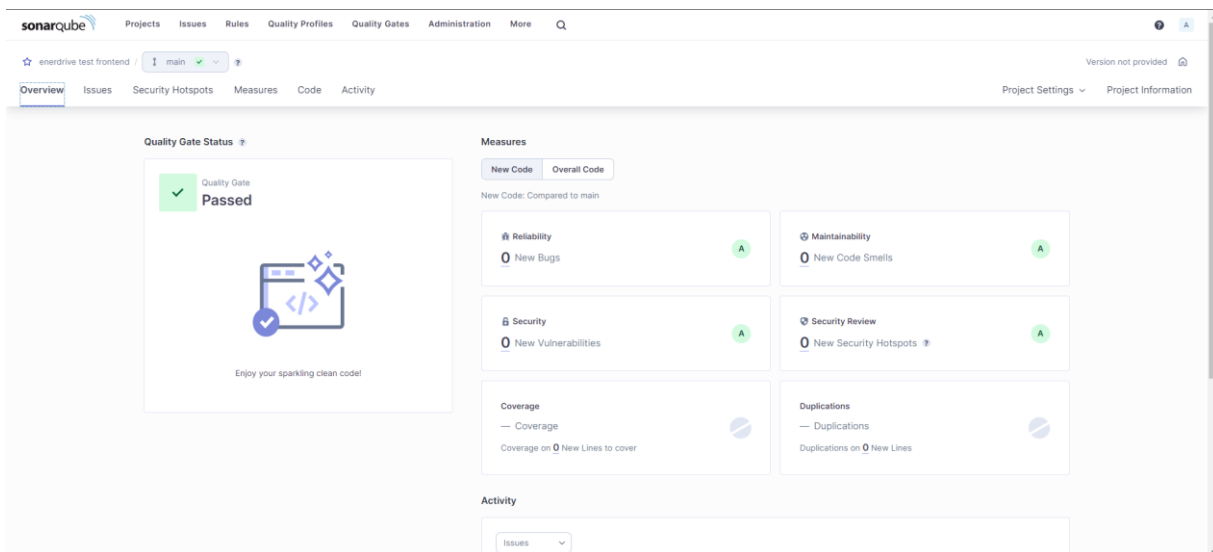
## TEST POUR LE SERVICE DE CONSOMMATION

L'évaluation complète par SonarQube permet de détecter les zones potentielles d'amélioration du code du service de consommation de carburant, en alignant le code sur les meilleures pratiques de développement logiciel.

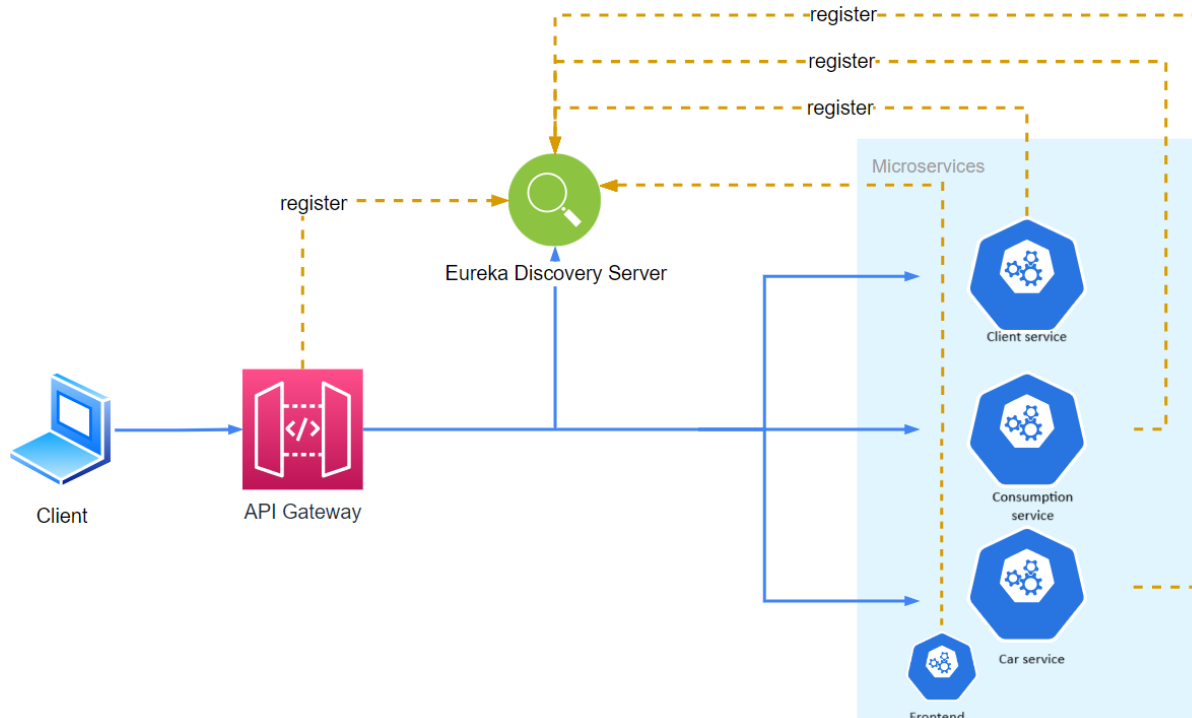


## TEST POUR LE FRONT-END

L'analyse SonarQube de la partie frontend vise à améliorer la qualité du code JavaScript et JSX, à garantir la cohérence et la lisibilité, et à identifier les zones nécessitant une attention particulière en termes de performance, de sécurité et de maintenabilité. Les résultats de cette analyse aident les développeurs à optimiser la qualité du frontend de l'application ReactJS



# ARCHITECTURE D'APPLICATION



Notre application contient 6 services suivants :

- Service client
- Service consommation
- Service voiture
- Service pour la partie front end
- Service api Gateway
- Service Eureka

**Services Backend (client, consommation, voiture) :** Ce sont les composants de l'application qui gèrent la logique métier, accèdent aux données, et fournissent des fonctionnalités spécifiques. Ils sont construits en utilisant Spring Boot, qui offre un cadre de développement robuste pour créer des applications Java cote backend. Ces services exposent souvent des API REST ou des points de terminaison pour permettre aux autres parties de l'application de communiquer avec eux.

**Service Frontend :** C'est la partie de l'application avec laquelle les utilisateurs interagissent directement. Dans une application web, il est développé en utilisant React,. Le service frontend communique avec les services backend pour récupérer et afficher des données, ainsi que pour envoyer des requêtes pour effectuer des actions.

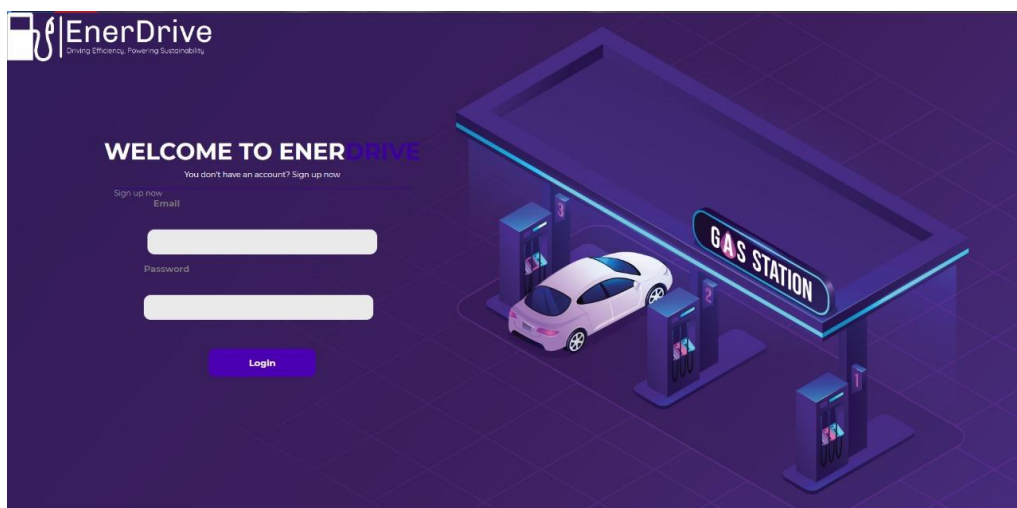
**API Gateway :** L'API Gateway est un composant central dans une architecture microservices qui agit comme une porte d'entrée pour toutes les requêtes client. Il fournit une interface unifiée pour accéder aux différents services backend de l'application. L'API Gateway peut également gérer des fonctionnalités telles que l'authentification, l'autorisation, la mise en cache

et la limitation du débit. Dans un écosystème Spring Boot, l'API Gateway être mis en œuvre à l'aide de Spring Cloud Gateway.

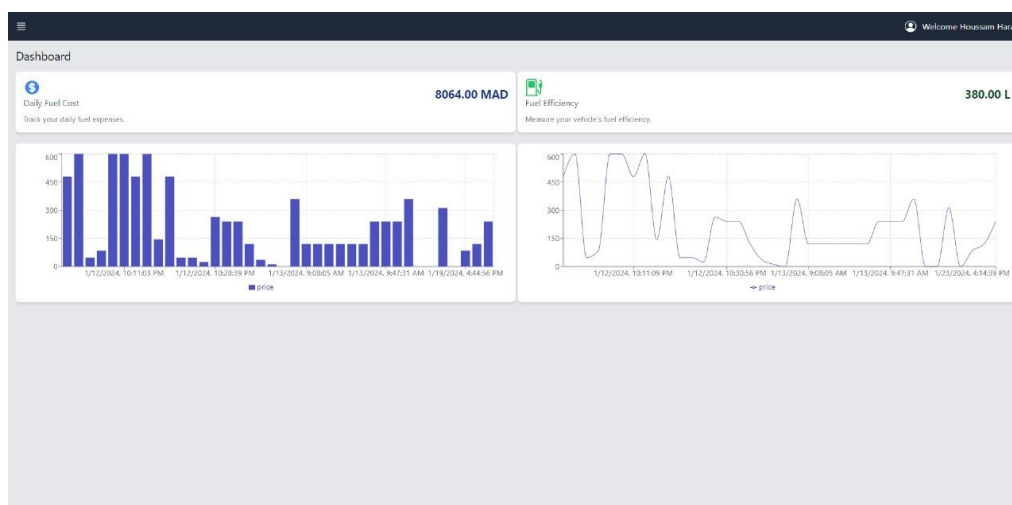
**Eureka Server :** Eureka Server est un serveur de registre de service dans le cadre de Spring Cloud Netflix, qui permet aux différents services d'enregistrer leur emplacement et aux clients de découvrir dynamiquement les services disponibles. Chaque service backend enregistré auprès du serveur Eureka est attribué à un nom logique, ce qui permet aux autres services de le trouver en utilisant ce nom plutôt qu'une URL statique. Cela facilite le développement d'applications distribuées et la mise à l'échelle horizontale des services.

## REALISATION ET INTERFACES

### LOGIN




### DASHBOARD




# CARS

Welcome Houssem Haraf


Search...




**Audi**  
A3  
Audi A3 2020  
[Add Car](#)




**Mercedes**  
Class E  
Mercedes Class E 2018  
[Add Car](#)




**BMW**  
Serie 2  
BMW Serie 2  
[Add Car](#)




**BMW**  
Serie 3  
BMW Serie 3  
[Add Car](#)





**BMW**  
Serie 4  
BMW Serie 4  
[Add Car](#)



**BMW**  
Serie 5  
BMW Serie 5  
[Add Car](#)














# INFORMATIONS

Welcome Houssem Haraf

Search...

PICTURE	NAME	RESERVE	CONSUMPTION/KM	START/STOP	FUEL	ACTIONS
	Audi A3	47L	10%	<input type="checkbox"/>	<div><div></div><div>0</div><div><a href="#">Fuel</a></div></div>	<a href="#">Delete</a>
	Mercedes Class E	44L	8%	<input type="checkbox"/>	<div><div></div><div>0</div><div><a href="#">Fuel</a></div></div>	<a href="#">Delete</a>
	BMW Serie 2	41L	9%	<input type="checkbox"/>	<div><div></div><div>0</div><div><a href="#">Fuel</a></div></div>	<a href="#">Delete</a>
	BMW Serie 7	18.3527L	10.3033%	<input type="checkbox"/>	<div><div></div><div>0</div><div><a href="#">Fuel</a></div></div>	<a href="#">Delete</a>
	BMW Serie 3	41.191L	7.33212%	<input type="checkbox"/>	<div><div></div><div>0</div><div><a href="#">Fuel</a></div></div>	<a href="#">Delete</a>
	BMW Serie 4	50L	7.51784%	<input type="checkbox"/>	<div><div></div><div>0</div><div><a href="#">Fuel</a></div></div>	<a href="#">Delete</a>
	BMW Serie X5	54.373L	10.1319%	<input type="checkbox"/>	<div><div></div><div>0</div><div><a href="#">Fuel</a></div></div>	<a href="#">Delete</a>

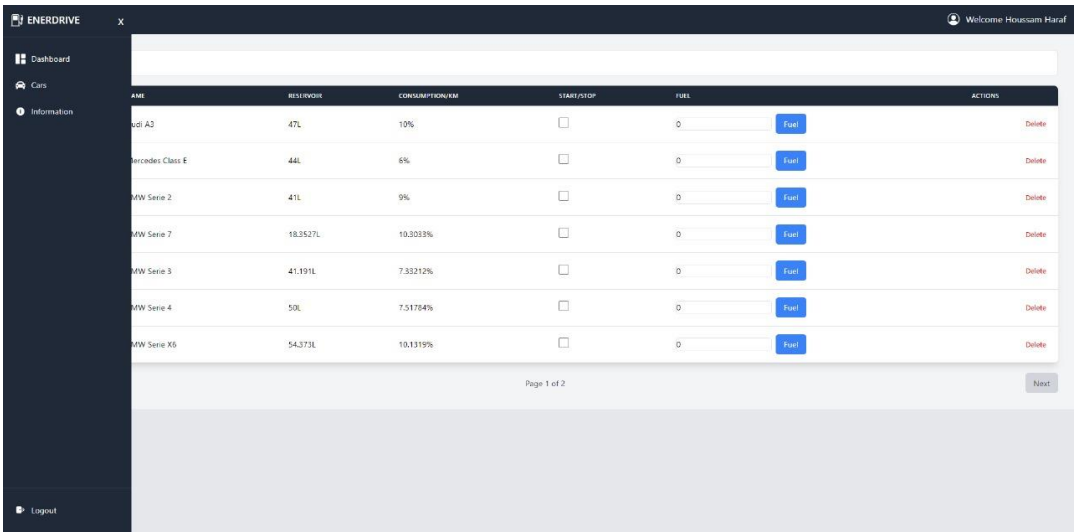
Previous

Page 1 of 2

Next

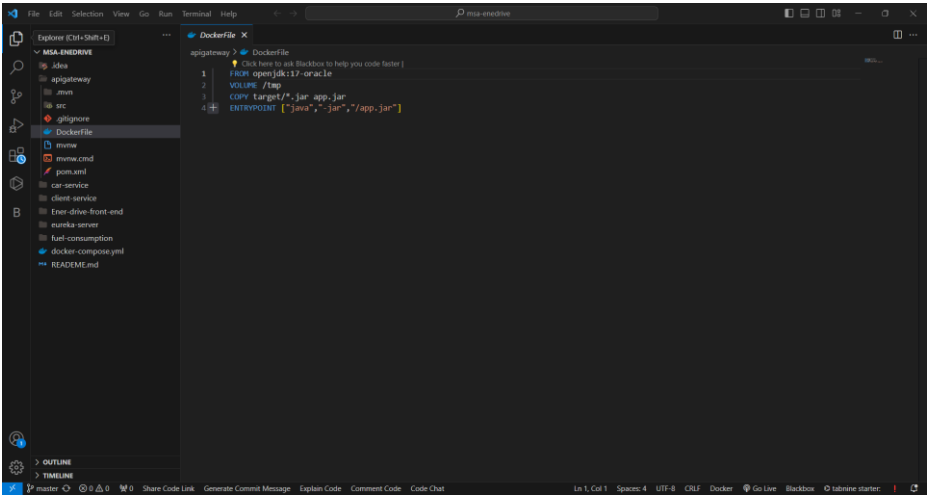
PAGE 12

# SIDEBAR MENU

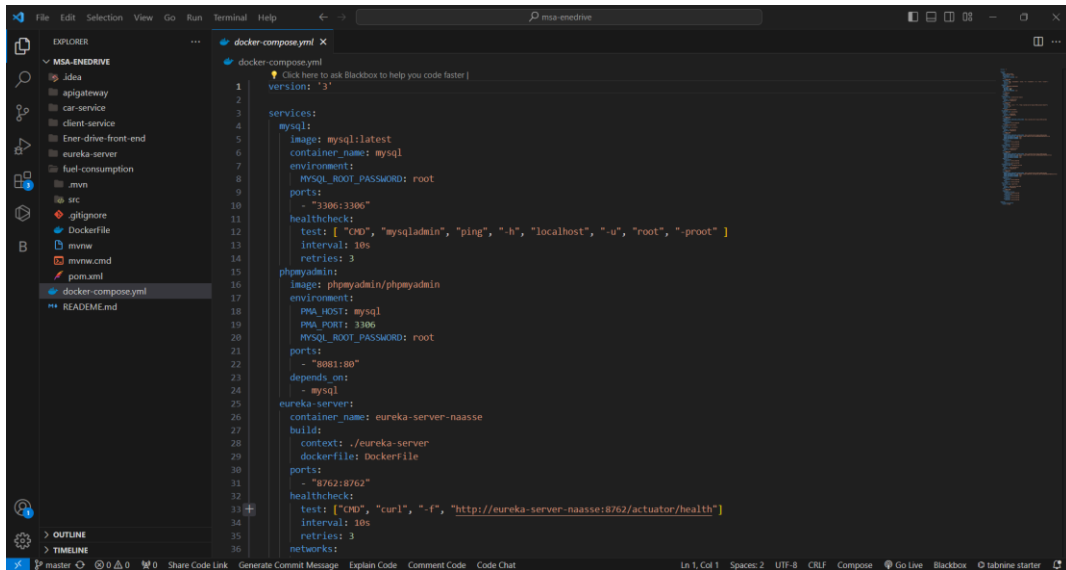


# DOCKER

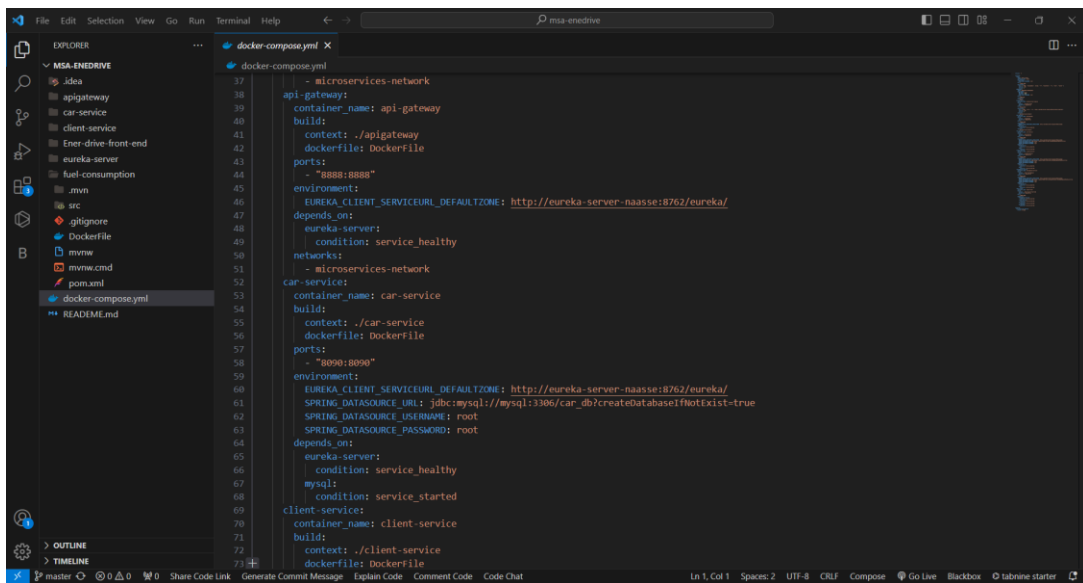
# DOCKER FILE



# DOCKER COMPOSE



```
1 version: '3'
2
3 services:
4   mysql:
5     image: mysql:latest
6     container_name: mysql
7     environment:
8       MYSQL_ROOT_PASSWORD: root
9     ports:
10      - "3306:3306"
11     healthcheck:
12       test: [ "CMD", "mysqladmin", "ping", "-h", "localhost", "-u", "root", "-proot" ]
13       interval: 10s
14       retries: 3
15   phpmyadmin:
16     image: phpmyadmin/phpmyadmin
17     environment:
18       PMA_HOST: mysql
19       PMA_PORT: 3306
20       MYSQL_ROOT_PASSWORD: root
21     ports:
22      - "8081:80"
23     depends_on:
24       - mysql
25   eureka-server:
26     container_name: eureka-server-naasse
27     build:
28       context: ./eureka-server
29       dockerfile: Dockerfile
30     ports:
31      - "8762:8762"
32     healthcheck:
33       test: [ "CMD", "curl", "-f", "http://eureka-server-naasse:8762/actuator/health" ]
34       interval: 10s
35       retries: 3
36     networks:
```



```
37 - microservices-network
38 api-gateway:
39   container_name: api-gateway
40   build:
41     context: ./apigateway
42     dockerfile: Dockerfile
43   ports:
44     - "8888:8888"
45   environment:
46     EUREKA_CLIENT_SERVICEURL_DEFAULTZONE: http://eureka-server-naasse:8762/eureka/
47   depends_on:
48     eureka-server:
49       condition: service_healthy
50   networks:
51     - microservices-network
52   car-service:
53     container_name: car-service
54     build:
55       context: ./car-service
56       dockerfile: Dockerfile
57     ports:
58       - "8090:8090"
59     environment:
60       EUREKA_CLIENT_SERVICEURL_DEFAULTZONE: http://eureka-server-naasse:8762/eureka/
61       SPRING_DATASOURCE_URL: jdbc:mysql://mysql:3306/car_db?createDatabaseIfNotExist=true
62       SPRING_DATASOURCE_USERNAME: root
63       SPRING_DATASOURCE_PASSWORD: root
64     depends_on:
65       eureka-server:
66         condition: service_healthy
67       mysql:
68         condition: service_started
69   client-service:
70     container_name: client-service
71     build:
72       context: ./client-service
73     dockerfile: Dockerfile
```



```

74 ports:
75   - "8099:8099"
76 environment:
77   EUREKA_CLIENT_SERVICEURL_DEFAULTZONE: http://eureka-server-naasse:8762/eureka/
78   SPRING_DATASOURCE_URL: jdbc:mysql://mysql:3306/client_db?createDatabaseIfNotExist=true
79   SPRING_DATASOURCE_USERNAME: root
80   SPRING_DATASOURCE_PASSWORD: root
81 depends_on:
82   eureka-server:
83     condition: service_healthy
84   mysql:
85     condition: service_started
86 consumption-service:
87   container_name: consumption-service
88   build:
89     context: ../fuel-consumption
90     dockerfile: Dockerfile
91 ports:
92   - "8083:8083"
93 environment:
94   EUREKA_CLIENT_SERVICEURL_DEFAULTZONE: http://eureka-server-naasse:8762/eureka/
95   SPRING_DATASOURCE_URL: jdbc:mysql://mysql:3306/fuel_consumption_db?createDatabaseIfNotExist=true
96   SPRING_DATASOURCE_USERNAME: root
97   SPRING_DATASOURCE_PASSWORD: root
98 depends_on:
99   eureka-server:
100     condition: service_healthy
101   mysql:
102     condition: service_started
103 react-front:
104   container_name: react-front
105   build:
106     context: ../ener-drive-front-end
107     dockerfile: Dockerfile
108 ports:
109   - "3000:3000"
110 depends_on:

```

```

110 depends_on:
111   consumption-service:
112     condition: service_started
113   client-service:
114     condition: service_started
115   car-service:
116     condition: service_started
117   api-gateway:
118     condition: service_started
119 networks:
120   microservices-network:
121     driver: bridge

```

# JENKINS

Tableau de bord > enedrive

Status: ✔ enedrive

Changements

Lancer un build

Configurer

Supprimer Pipeline

Full Stage View

GitHub

SonarQube

SonarQube

SonarQube

SonarQube

Réorganiser

Pipeline Syntax

GitHub Hook Log

Historique des builds

tendance

Ajouter une description

Désactiver le projet

Stage View

	Declarative: Tool Install	Git Clone	Eureka Server Build	Eureka SonarQube	Gateway Build	Gateway SonarQube	Car Service Build	CAR Sonar	Client Service Build	Client SonarQube	Fuel Service Build	Fuel Service SonarQube Analysis	Create Docker Compose
Average stage times: (Average full run time: ~10min)	174ms	3s	39s	54s	37s	55s	47s	1min 7s	47s	1min 5s	44s	1min 2s	30s
jenkins-20 06:02 1 comment	147ms	3s	47s	55s	33s	49s	57s	1min 3s	50s	1min 11s	47s	1min 2s	1min 55s
jenkins-20 05:16 2 comments	176ms	2s	35s	46s	37s	1min 5s	1min 1s	1min 24s	1min 4s	1min 37s	1min 2s	1min 27s	12s
jenkins-20 05:02 6 comments	168ms	3s	39s	1min 3s	43s	1min 4s	1min 0s	1min 28s	1min 6s	1min 37s	1min 3s	1min 29s	10s