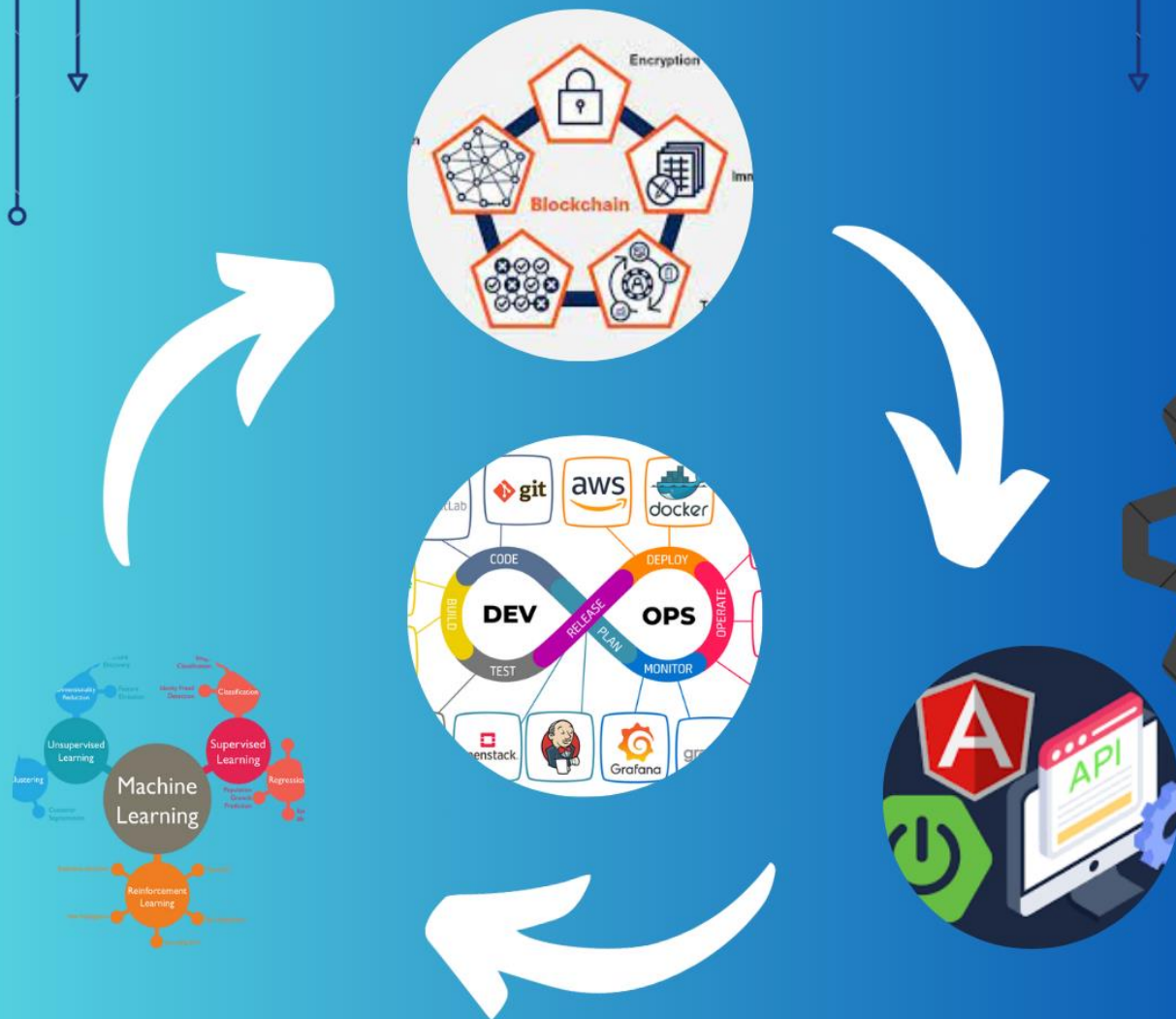


RAPPORT PROJET FIN DU MODULE : ML/JEE



Encadré par :Lotfi EL AACHAK

Realisé par: BECHARRI Hajar

EL MACHKOURI Imane

Essalhi Sara

EL KAISSI Houda

Table des matières

<u>I.Introduction</u>	4
<u>II.Developpement</u>	5
<u>1.explanation ode Spring Boot and Angular</u>	5
<u>3.L'architecture de microservices</u>	5
<u>Microservice est un style architectural qui structure une application comme une collection de services indépendants, déployables de manière autonome. Au lieu de construire une application monolithique unique, les microservices permettent de diviser une application en composants plus petits et autonomes, appelés microservices,</u>	5
<u>Principes Fondamentaux</u>	5
<u>4.Communication entre Microservices :</u>	6
<u>5.envirement de travail</u>	7
<u>6.technologies utilisé</u>	8
<u>7.Frontend:angular</u>	9
<u>8.Organisation de frontend</u>	10
<u>9.Backend:springboot</u>	12
<u>10.Conception :</u>	12
<u>11.interface</u>	14
<u>II.Devops</u>	18
<u>1.Definition devops :</u>	18
<u>2.Les principe de devops :</u>	19
<u>1.Les outils utilise dans le projet :</u>	20
<u>Docker :</u>	21
<u>Jenkins :</u>	21
<u>Kubernetes (avec Minikube) :</u>	22
<u>8.Deploiment du MachineLearning :</u>	30
<u>8.La base de donne :</u>	31
<u>PersistentVolumeClaim (PVC) :</u>	31
<u>Deployment :</u>	32
<u>Service :</u>	33
<u>ConfigMap :</u>	34
<u>Secret :</u>	34
<u>III.Machine learning</u>	35
<u>1.Collecte de Données :</u>	35
<u>2.Prétraitement de Données :</u>	35

<u>3.Reduction du Dimension (t-SNE, PCA) :</u>	36
<u>4.Clustering (EM) :</u>	38
<u>5.Le nombre de Clusters K déterminé :</u>	40
<u>6.Modèle de Classification (RF) :</u>	40
<u>7.Evaluation des trois modèles :</u>	40
<u>IV.Blockchain : Gestion d'Assurance et de Réclamations</u>	42
<u>1.Technologies Utilisées</u>	42
<u>2.Contrats Intelligents</u>	43
<u>3.Intégration Blockchain avec Angular.</u>	45
<u>4.Fichiers de Configuration.</u>	49
<u>V.Conclusion</u>	52
<u>VI.Remerciements</u>	53

I. Introduction

Dans le paysage dynamique du secteur des assurances, la gestion efficace des contrats est cruciale pour garantir la transparence, la précision et la rentabilité. Dans cette optique, notre projet vise à révolutionner la gestion des contrats d'assurance en intégrant des smart contrats, offrant ainsi une alternative plus transparente et économique. L'application que nous développons sera centrée sur le suivi détaillé des contrats d'assurance, avec une mise en œuvre robuste de la gestion du métier d'assurance dans le backend Spring Boot.

La transformation numérique du secteur des assurances nécessite une approche holistique, allant au-delà de la simple gestion des contrats. Ainsi, notre projet s'étend également à l'implémentation d'un système de recommandation avancé basé sur des algorithmes de machine learning. Ce système aura pour objectif de conseiller aux clients des offres d'assurance correspondant au mieux à leurs profils, offrant ainsi une expérience personnalisée et répondant aux besoins spécifiques de chaque individu.

La mise en place de cette solution innovante implique l'utilisation d'outils et de pratiques de pointe dans le domaine du développement logiciel et de l'apprentissage automatique. Le backend sera construit en utilisant Spring Boot, tandis que les principaux outils DevOps tels que Git, Docker, Kubernetes et Jenkins seront intégrés pour assurer une gestion efficace du cycle de vie de l'application. Par ailleurs, la mise en œuvre du système de recommandation se fera à travers Flask, en exploitant les capacités des outils MLOps tels que Git, Docker, AirFlow et Kafka.

Le déploiement et la gestion des modèles de machine learning seront facilités grâce à l'utilisation de Kubeflow pour la mise en place du pipeline. Cette approche MLOps garantira la cohérence, la reproductibilité et la scalabilité des modèles, tout en permettant une gestion optimale du cycle de vie des algorithmes.

En somme, notre projet aspire à élever la gestion des contrats d'assurance vers de nouveaux sommets en intégrant des technologies émergentes, des pratiques DevOps et des algorithmes de machine learning. À travers cette initiative, nous visons à offrir une solution complète et novatrice qui redéfinira la manière dont les entreprises d'assurance interagissent avec leurs clients, tout en améliorant l'efficacité opérationnelle globale du secteur.

II. Développement

1. explication ode Spring Boot and Angular

spring boot :pring Boot est un projet du framework Spring qui simplifie considérablement le développement d'applications Java. Il fournit un ensemble d'outils et de conventions par défaut pour créer des applications autonomes, prêtes à être déployées. Avec Spring Boot, vous pouvez créer rapidement des applications

angular :Angular est un framework open source développé par Google pour la création d'applications web dynamiques et single-page (SPA). Il permet de construire des interfaces utilisateur interactives en utilisant le langage TypeScript. Angular suit le modèle de conception MVC (Model-View-Controller) et offre des fonctionnalités telles que la liaison de données bidirectionnelle, l'injection de dépendances, et une architecture modulaire.

3. L'architecture de microservices

Microservice est un style architectural qui structure une application comme une collection de services indépendants, déployables de manière autonome. Au lieu de construire une application monolithique unique, les microservices permettent de diviser une application en composants plus petits et autonomes, appelés microservices,

Principes Fondamentaux

Indépendance des Services :

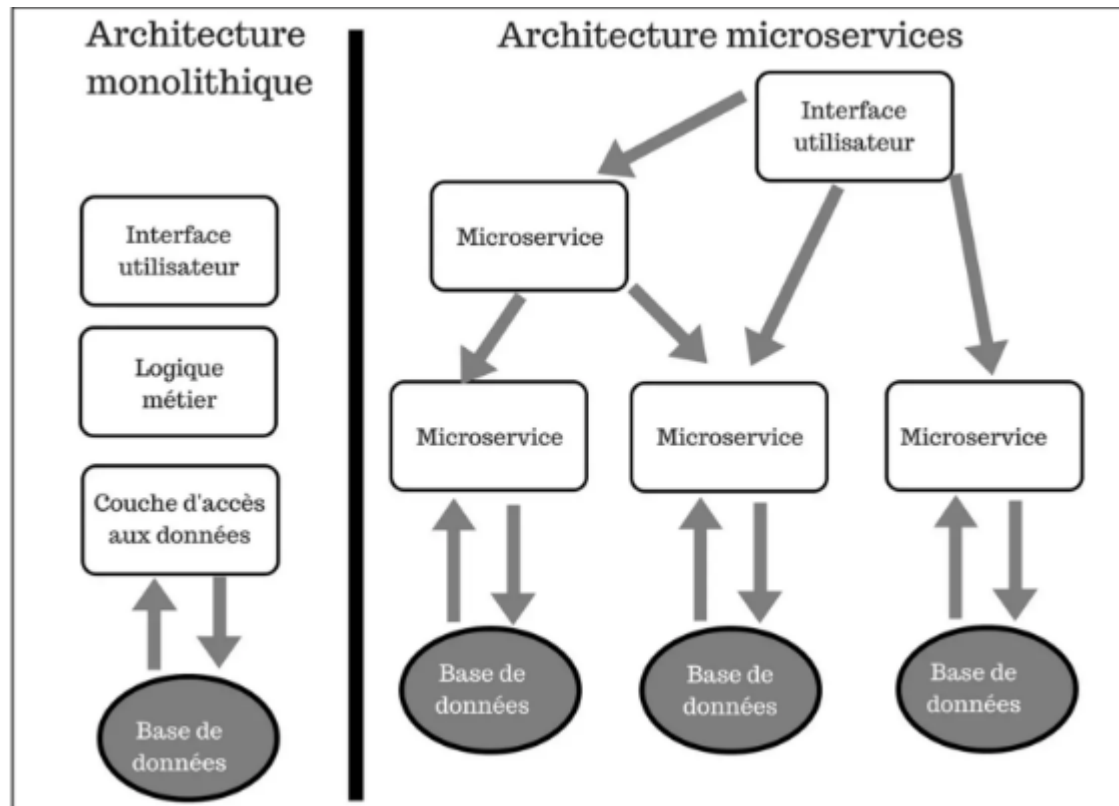
Chaque microservice est indépendant et peut être développé, déployé et évolué de manière autonome.

Communication légère

Les microservices communiquent entre eux via des protocoles légers tels que HTTP/REST ou des mécanismes asynchrones.

GestionDécentralisée des données

Chaque microservice gère ses propres données, favorisant ainsi l'indépendance et la scalabilité. Les microservices exposent des API (Interfaces de Programmation Applicative) pour permettre la communication avec d'autres microservices. Ces API définissent les contrats entre les services.



4. Communication entre Microservices :

Protocoles de communication: Les microservices communiquent généralement via des protocoles légers tels que HTTP/REST, gRPC, ou des mécanismes asynchrones tels que les messages.

Api gateway: Une API Gateway peut être utilisée pour centraliser les points d'entrée et gérer les requêtes/réponses entre les clients et les microservices, facilitant la gestion des communications.

Echanges asynchrones: Certains scénarios nécessitent une communication asynchrone, où un microservice envoie un message à un autre et continue son travail sans attendre de réponse immédiate.

Gestion des erreurs:

Les mécanismes de gestion des erreurs et de reprise sur erreur sont cruciaux pour garantir la robustesse du système, compte tenu de la distribution des services.

6. Stockage de Données et Choix de Base de Données :

Bases de Bases de données Décentralisées: Chaque microservice gère généralement sa propre base de données, favorisant l'indépendance et la scalabilité. Les bases de données peuvent être relationnelles, NoSQL, ou adaptées à des besoins spécifiques.

Consistance des données: Le maintien de la cohérence des données entre les microservices peut être un défi. Des modèles de cohérence tels que l'événementiel ou la compensation peuvent être utilisés.

Stockage des données temporaires: Certains microservices peuvent nécessiter un stockage temporaire de données, qui peut être géré par des caches distribués ou des bases de données in-memory.

Gestion des migrations des données: Des mécanismes pour gérer les migrations de données entre les microservices et les versions de bases de données sont nécessaires pour maintenir la stabilité.

5. environnement de travail

intellij

IntelliJ IDEA est un environnement de développement intégré (IDE) très populaire pour la programmation Java, mais il prend également en charge d'autres langages tels que Kotlin, Groovy, Scala, et plus encore. Voici un guide rapide sur la configuration de votre environnement de travail dans IntelliJ IDEA

postman

Postman est un outil populaire utilisé par les développeurs pour tester, développer et documenter des API. Il offre une interface conviviale qui permet d'envoyer des requêtes HTTP à des API et de visualiser les réponses. Voici une introduction à l'utilisation de Postman :

mysql

MySQL est un système de gestion de base de données relationnelle (SGBDR) open source qui est largement utilisé dans le développement d'applications web et d'autres applications nécessitant la gestion de données. Voici une brève introduction à MySQL :

Angular

est un framework open source développé et maintenu par Google. Il est utilisé pour la création d'applications web dynamiques et interactives du côté client. Angular est écrit en TypeScript et suit le modèle de conception MVC (Modèle-Vue-Contrôleur). Voici une brève introduction à Angular :

6. technologies utilisé

Typescript: Extension de JavaScript, TypeScript introduit un typage statique améliorant la qualité et la maintenance du code, en détectant les erreurs tôt dans le processus de développement, principalement adapté aux grands projets.:

Angular materiel:

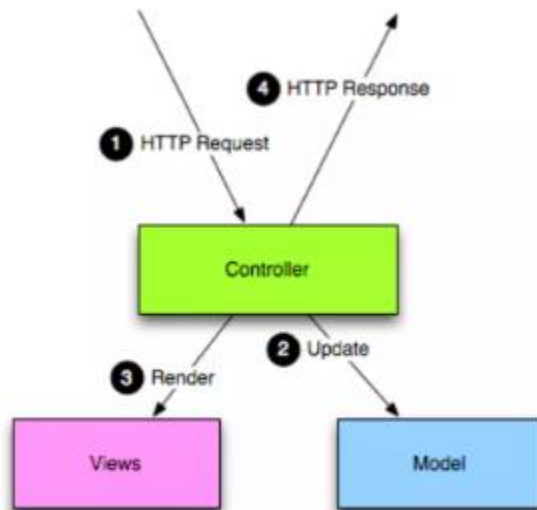
Bibliothèque de composants pour Angular, Angular Material suit les principes du design Material, offrant des composants pré-construits pour des interfaces utilisateur cohérentes et attrayantes. Elle simplifie le développement d'applications Angular en garantissant une apparence professionnelle et conviviale.

Api Resful: Un style architectural standard pour concevoir des applications réseau, où les microservices Spring Boot exposent des API RESTful consommées par Angular.

HTTPS/HTTP: Protocoles de communication standard pour les échanges entre le frontend et le backend, assurant le transfert de données sur le web.

JSON web Tokens Un moyen sécurisé et compact de transférer des revendications entre le frontend et le backend, souvent utilisé pour l'authentification et l'autorisation des utilisateurs.

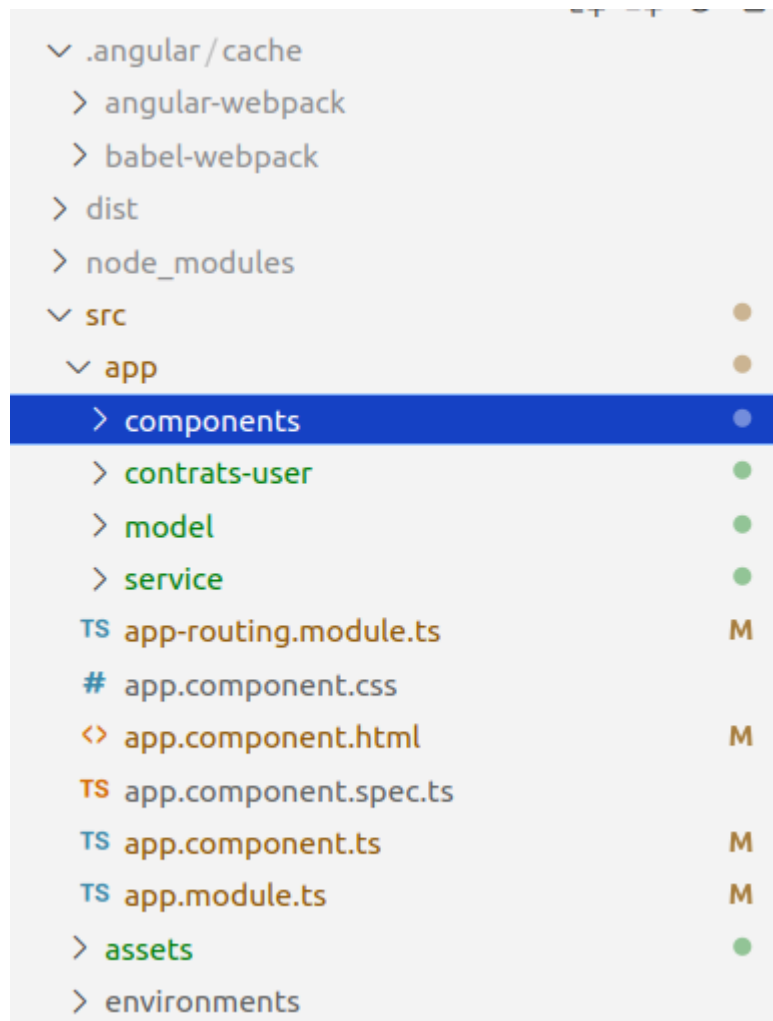
MVC



7. Frontend:angular

Fontend La conception et la mise en place du frontend est la partie que j'ai mené entièrement. C'est un travail où j'ai donc investi le plus de temps pour apprendre et approfondir. Les technologies du backend et de la base de données, ont été choisies à partir des projets précédents, mais pour la partie frontend, j'avais tout a réalisé avec des technologies nouvelles.

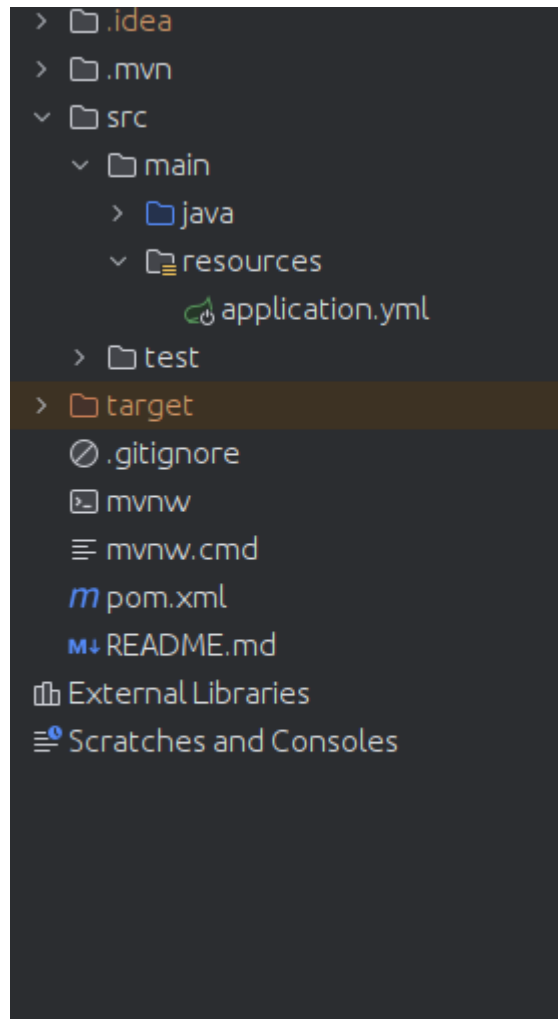
8. Organisation de frontend





node_module, package.json : les packets et modules des dépendances des framework javascript pour le frontend. • tslint.json, tsconfig.json : Fichiers de configuration pour TypeScript. • angular-cli.json : configuration globale pour angular cli – une « command line interface » pour construire des applications Angular 4 en utilisant le style nodejs. • Dockerfile, docker-compose : Configuration pour docker. • src : comme le backend et la base de donnée, l'organisation des répertoires du frontend est structurée avec deux principaux répertoires, celui pour le core et celui pour les modules. Le core contient des composants communs à l'ensemble de GeoNature, et les modules contiennent des modules séparés correspondant à chaque protocole.

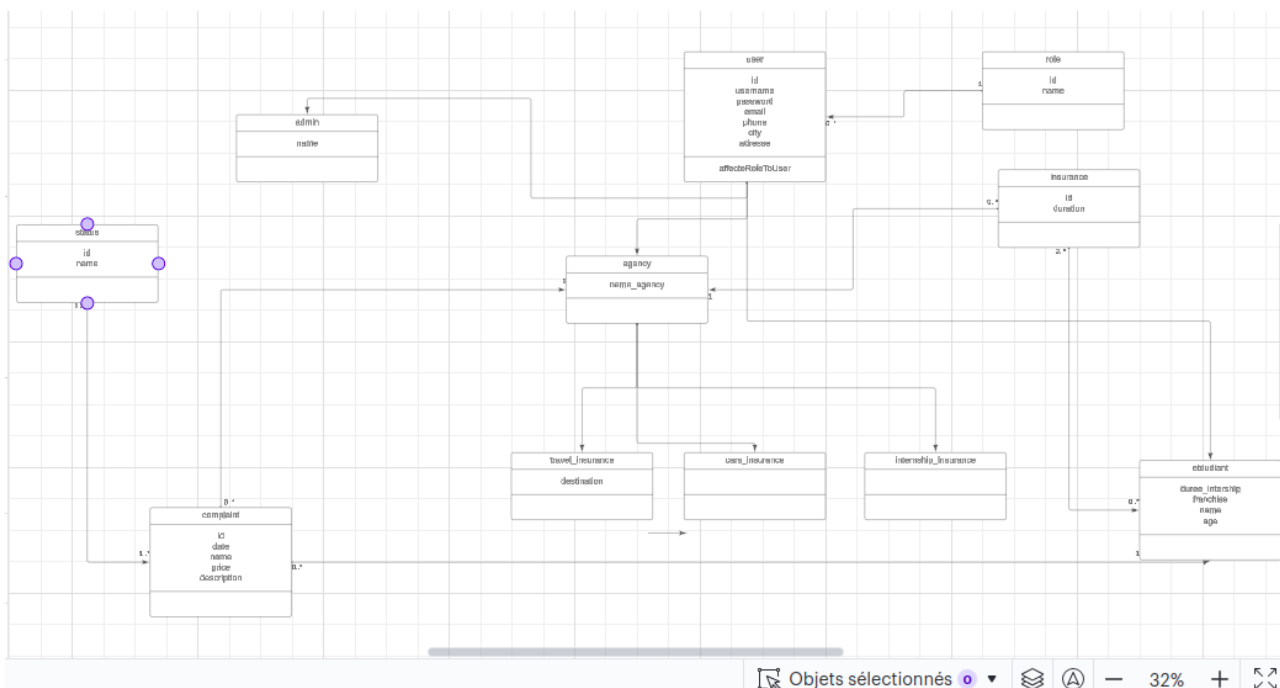
9. Backend:springboot



10. Conception :

diagramme de classe

Un diagramme de classe est un type de diagramme de modélisation UML (Unified Modeling Language) qui représente les classes et les relations entre ces classes. Il est largement utilisé pour visualiser la structure statique d'un système logiciel.



11. interface

login form :

Login

Email

houda@gmail.com

Password

Sign Up

register
form :

Username:

a

Password:

Nom:

Prenom:

Telephone:

Role:

Account Address:

Confirm Password:

home
page de
client

← → ↺

localhost:4200/dashboard

🔍

🔗

🌟

⚙️

📱

🛑 En pause

Redémarrer pour mettre à jour ⚙️

Assurance App

Register Login Search for Insurance Logout Add reclamation Show Contracts in Cart

CONTRACTS:

9999999

Sort by CostSort by Date

Cost:

Description:

Date:

More Details ADD TO CART

Cost:

100222222222220

Description:

Date:

2024-01-22

More Details ADD TO CART

Cost:

100222222222220

Description:

Date:

2024-01-22

More Details ADD TO CART

Cost:

100222222222220

Description:

Date:

2024-01-22

More Details ADD TO CART

Cost:

4

Description:

Date:

houdakaissi17@gmail.com

More Details ADD TO CART

Cost:

4

Description:

Date:

houdakaissi17@gmail.com

More Details ADD TO CART

Cost:

88

Description:

Date:

houdakaissi17@gmail.com

More Details ADD TO CART

Cost:

88

Description:

Date:

houdakaissi17@gmail.com

More Details ADD TO CART

Cost:

88


Description:

Date:

houdakaissi17@gmail.com

More Details ADD TO CART

chercher assurances:form

 Assurance App

Do you have a car?

☐ Yes

☐ No

Miliage

Age:

Do you have a stage?

☐ Yes

☐ No

Duration

Start Franchise:

End Franchise:

Do you want to travel?

☒ Yes

☐ No

Start Franchise:

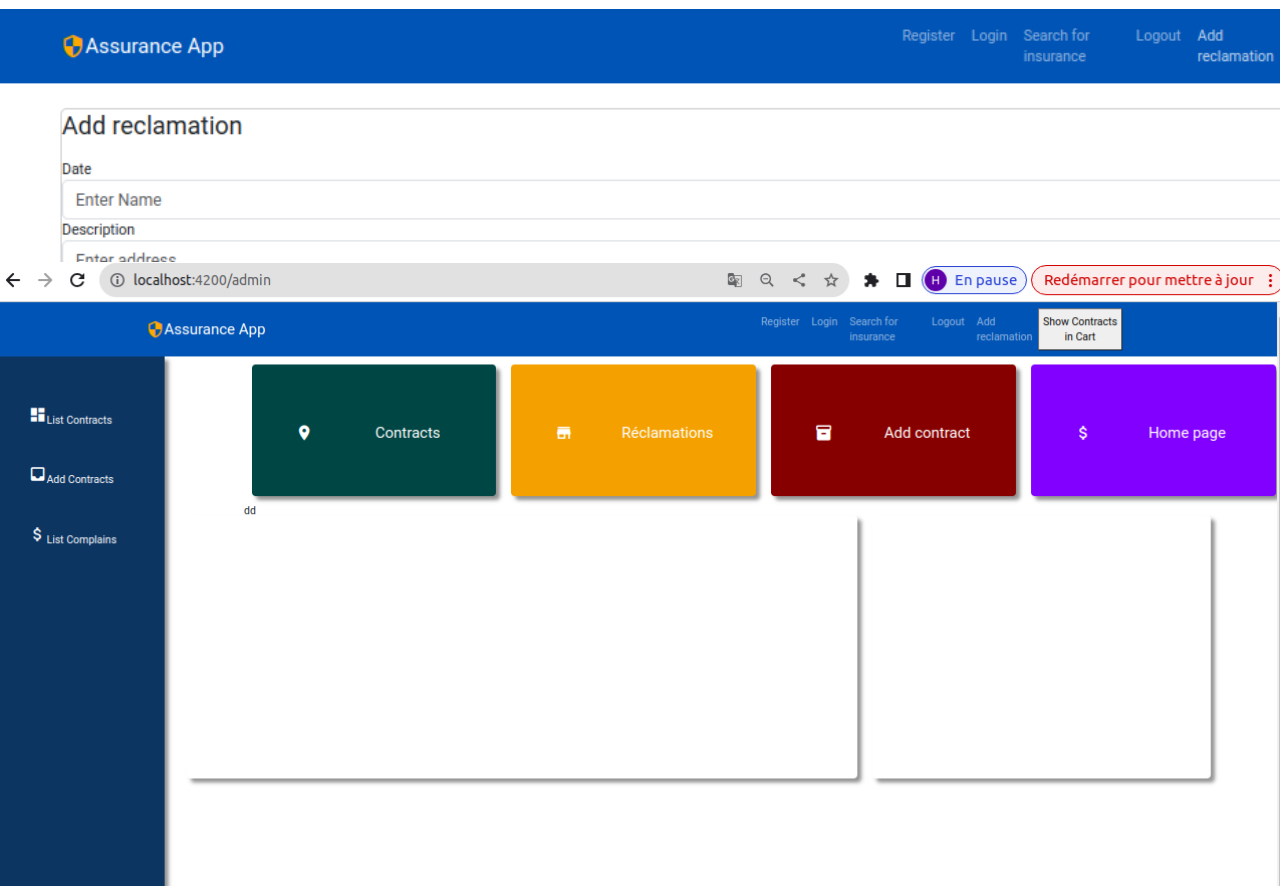
End Franchise

Duration

Destination


Submit


client : faire une reclamation




page de
entreprise

entreprise :ajouter une assurance

List Contracts

Add Contracts

List Complains

Add new contract

Username

Enter Name

password

Enter address

Cost of contract

Enter Mobile

Date

Enter Mobile

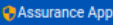
Description

Enter Mobile


Submit


lister ses


assurances



[Register](#) [Login](#) [Search for insurance](#) [Logout](#) [Add reclamation](#)

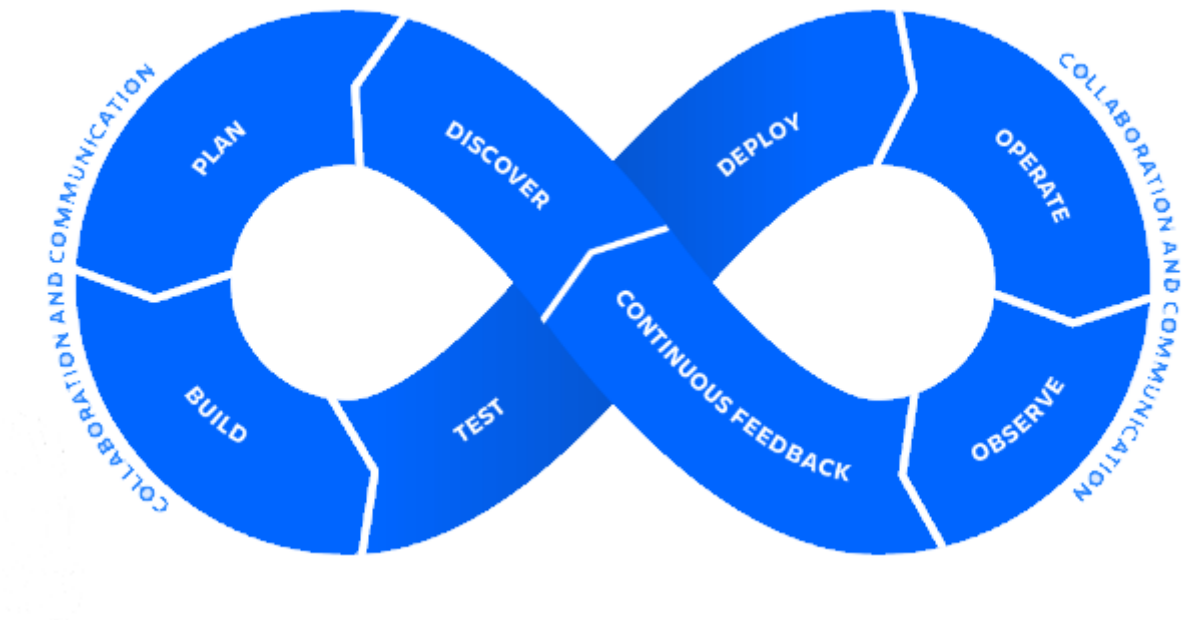
List Contracts

Add Contracts

List Complains

Cost assurance	DescriptionDate of assurance	seller username
100222222222220	2024-01-22	a
100222222222220	2024-01-22	a
100222222222220	2024-01-22	a
4	houdakaiss17@gmail.comfff ffff	
4	houdakaiss17@gmail.comfff ffff	
88	houdakaiss17@gmail.comfff ffff	
4	houdakaiss17@gmail.comfff ffff	
88	houdakaiss17@gmail.comyour_username	
4	22	your_usernameee

II. Devops



1. Definition devops :

DevOps est une approche de développement logiciel qui vise à intégrer étroitement le développement et l'exploitation (ops) dans le but d'améliorer l'efficacité et la qualité du cycle de vie du développement logiciel. Le terme "DevOps" est une contraction des mots "Development" (développement) et "Operations" (opérations).

L'objectif principal de DevOps est de favoriser la collaboration et la communication entre les équipes de développement et d'exploitation, traditionnellement considérées comme distinctes. Cela se fait en automatisant les processus de développement, de test et de déploiement, en mettant en œuvre des pratiques de gestion de configuration, et en utilisant des outils de surveillance et de gestion des logs.

DevOps vise à créer un cycle de développement continu (Continuous Integration - CI) et un déploiement continu (Continuous Deployment - CD), permettant aux équipes de livrer des logiciels de manière plus rapide, fiable et efficace. En adoptant les principes de DevOps, les organisations cherchent à réduire les silos entre les équipes, à accélérer le time-to-market, à améliorer la qualité des logiciels et à répondre plus rapidement aux besoins des utilisateurs finaux.

2. Les principe de devops :

DevOps englobe un ensemble de pratiques, de principes et d'outils visant à améliorer la collaboration entre les équipes de développement et d'exploitation. Les éléments clés intégrés dans DevOps comprennent :

1. **Collaboration et communication** : DevOps met l'accent sur la collaboration étroite entre les équipes de développement, d'exploitation et d'autres parties prenantes. La communication transparente est essentielle pour surmonter les silos traditionnels.
2. **Automatisation** : L'automatisation est au cœur de DevOps. Elle concerne l'automatisation des processus de développement, de tests, de déploiement, de gestion de configuration, et d'autres tâches opérationnelles. Cela permet de réduire les erreurs humaines, d'accélérer les cycles de développement, et d'améliorer la cohérence.
3. **Intégration continue (CI)** : La CI consiste à intégrer fréquemment les changements de code dans un référentiel partagé. Cela permet de détecter rapidement les erreurs, d'assurer une cohérence constante et de faciliter le déploiement continu.
4. **Déploiement continu (CD)** : La CD étend la CI en automatisant le processus de déploiement des applications dans des environnements de production après chaque modification réussie. Cela garantit des déploiements plus rapides et fiables.
5. **Gestion de configuration** : DevOps utilise des outils de gestion de configuration pour garantir que les infrastructures, les plateformes et les environnements sont configurés de manière reproductible et cohérente.
6. **Monitoring et feedback** : La surveillance continue des performances, des logs et des métriques permet d'identifier rapidement les problèmes en production. Les retours d'information sont essentiels pour améliorer en permanence le processus de développement et d'exploitation.
7. **Infrastructure as Code (IaC)** : L'IaC permet de décrire et de provisionner l'infrastructure de manière automatisée en utilisant des scripts ou des configurations. Cela facilite la gestion et la reproduction des environnements.
8. **Sécurité** : L'intégration de la sécurité dès le début du cycle de vie du développement est un aspect important de DevOps. Les pratiques de DevSecOps intègrent la sécurité dans l'ensemble du processus.
9. **Culture DevOps** : Au-delà des pratiques et des outils, la culture DevOps encourage la responsabilité partagée, la confiance, la prise de risques mesurée, et la volonté d'apprendre et de s'améliorer constamment.

1. Les outils utilisés dans le projet :

Dans notre projet, nous avons intégré de manière synergique trois outils majeurs du domaine DevOps pour optimiser notre processus de développement et de déploiement. Nous avons utilisé Docker pour la création et la gestion efficace de conteneurs, Jenkins pour l'intégration continue (CI), et Kubernetes (via l'outil Minikube) pour l'orchestration et le déploiement continu (CD).

Docker :

- *Description* : Docker est une plateforme de conteneurisation qui offre une méthodologie d'empaquetage d'applications et de leurs dépendances dans des conteneurs isolés. Ces conteneurs garantissent une portabilité élevée et simplifient les processus de déploiement.
- *Utilisation dans notre projet* : Nous avons utilisé Docker pour créer des conteneurs, fournissant ainsi des environnements de développement cohérents, isolés et reproductibles.

Jenkins :

- *Description* : Jenkins est un serveur d'intégration continue open source, automatisant les étapes de construction, de test et de déploiement des applications. Il facilite la détection rapide d'erreurs et assure une intégration harmonieuse du code.
- *Utilisation dans notre projet* : Jenkins a été déployé en tant que composant essentiel de notre pipeline DevOps, automatisant les processus de CI. Il surveille les modifications de code, initie des builds automatiques, exécute des tests, et facilite le déploiement continu.

Kubernetes (Minikube) :

- *Description* : Kubernetes est un système d'orchestration de conteneurs, simplifiant le déploiement, la gestion, et la mise à l'échelle d'applications conteneurisées. Minikube, quant à lui, est une distribution légère de Kubernetes, adaptée à des environnements de développement et de test locaux.
- *Utilisation dans notre projet* : Nous avons choisi d'utiliser Minikube pour créer un cluster Kubernetes local. Cette configuration nous a permis de tester et de déployer nos applications de manière similaire à un environnement de production.

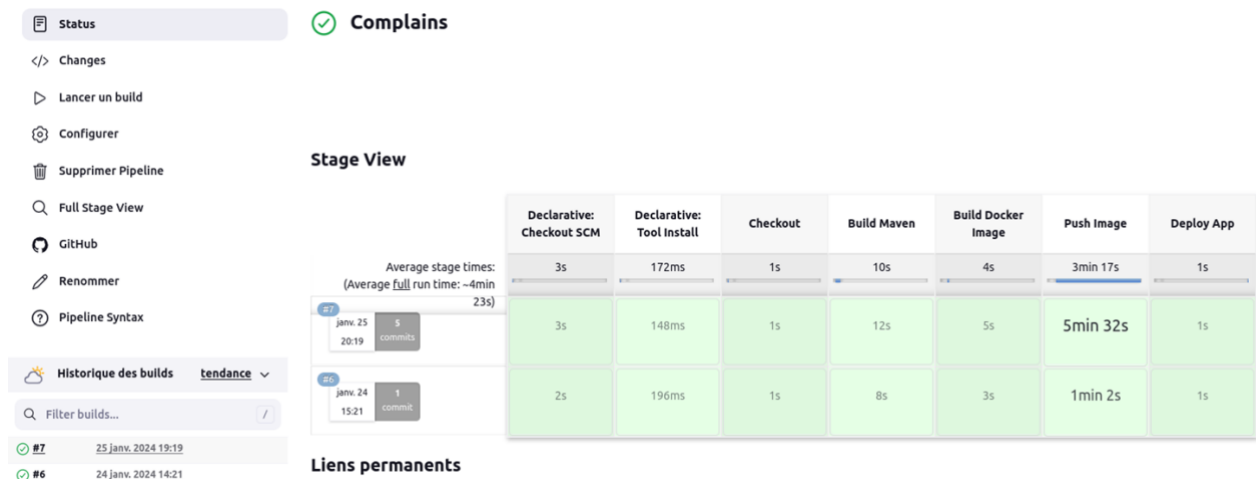
1. Les elements des outils devops :

Les éléments qui seront évoqués ci-dessous constituent la trame fondamentale des structures et des composants essentiels de Docker, Jenkins et Kubernetes. Leur exploration approfondie est impérative pour appréhender de manière exhaustive le fonctionnement et l'interaction de ces outils dans le contexte d'un environnement DevOps.

Docker :

1. **Image** : Une image Docker est un modèle léger et autonome qui comprend le code source, les bibliothèques, les dépendances, les variables d'environnement, et d'autres éléments nécessaires à l'exécution d'une application.
2. **Conteneur** : Un conteneur Docker est une instance en cours d'exécution d'une image. Les conteneurs sont isolés les uns des autres et de l'environnement hôte, permettant une portabilité et une cohérence accrues.
3. **Dockerfile** : Un fichier de configuration textuel qui définit les instructions nécessaires à la création d'une image Docker. Il décrit les étapes pour construire une image de conteneur.

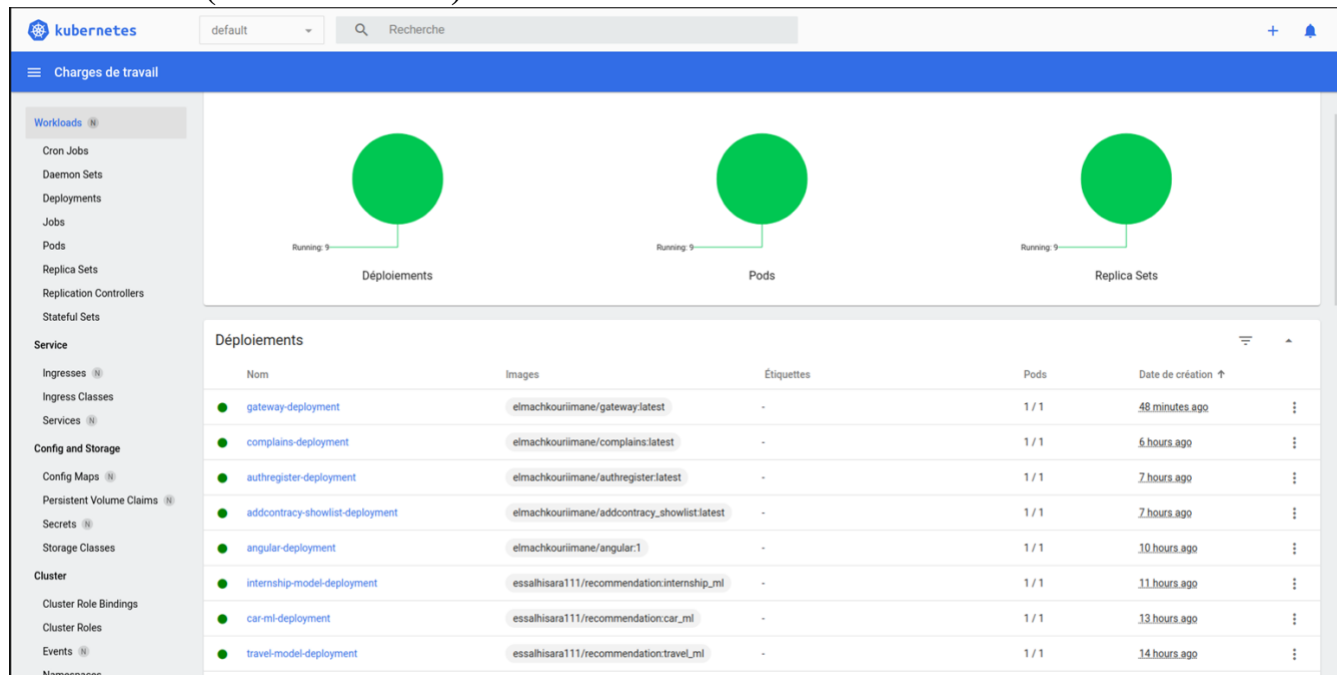
Jenkins :



1. **Job** : Un job Jenkins représente une tâche spécifique, telle que la construction, le test, ou le déploiement d'une application. Les jobs sont définis dans Jenkins pour automatiser différentes étapes du processus de développement.
2. **Pipeline** : Un pipeline Jenkins est une suite de jobs interconnectés qui décrivent le flux complet de travail, de l'intégration continue à la livraison continue.
3. **Jenkinsfile** : Un fichier textuel qui définit la configuration d'un pipeline Jenkins en tant que code. Il peut être stocké dans le gestionnaire de code source pour permettre une gestion du code en tant que code.
4. **Agent** : Un **agent Jenkins** est un composant logiciel qui permet la distribution de tâches de construction, de test et de déploiement sur des machines distantes. Il fonctionne en tant qu'entité

autonome, communique avec le serveur Jenkins principal, et exécute des travaux spécifiques de manière isolée. Cela contribue à l'évolutivité et à l'efficacité des environnements d'intégration continue.

Kubernetes (avec Minikube) :

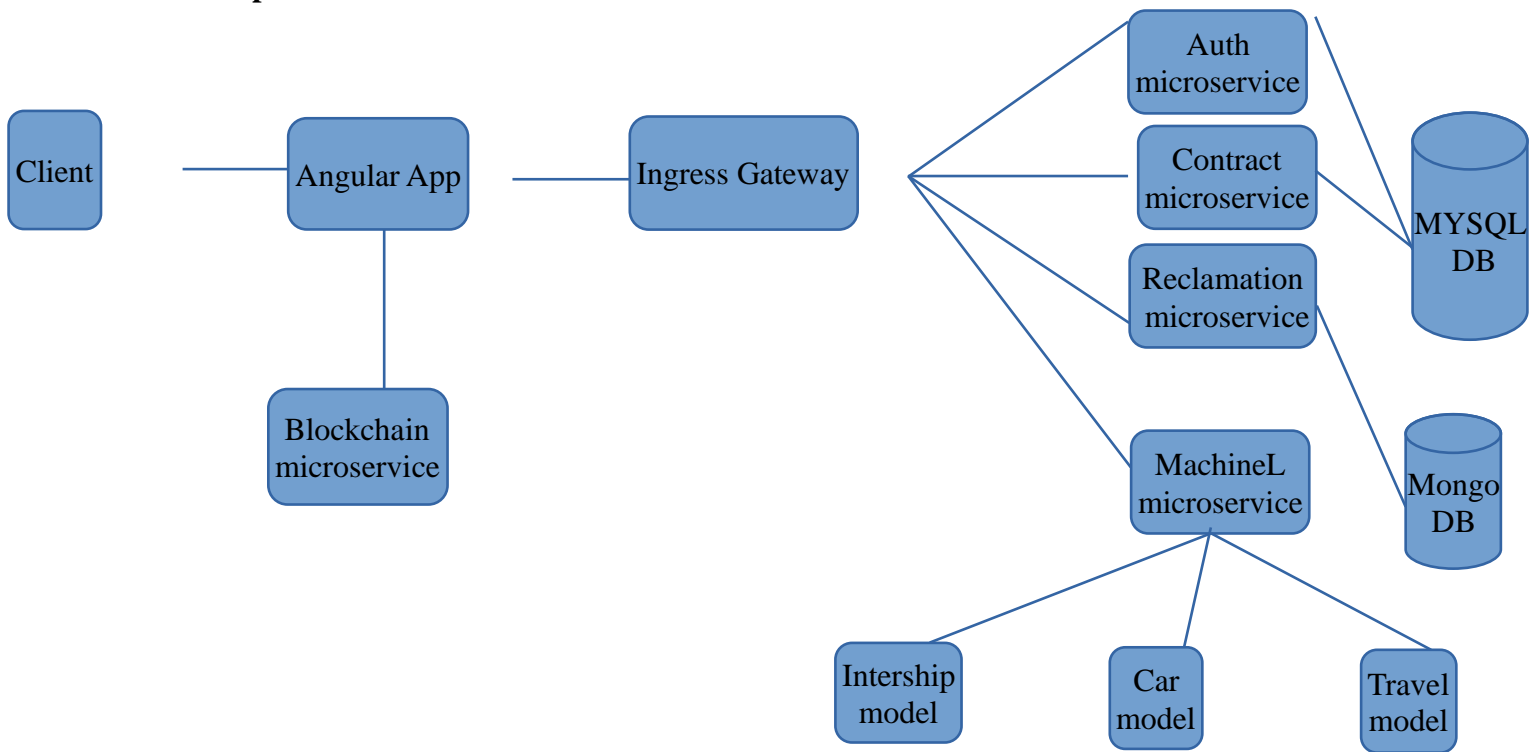


The screenshot shows the Kubernetes dashboard interface. At the top, there's a header with the 'kubernetes' logo, a dropdown menu set to 'default', and a search bar labeled 'Recherche'. Below the header, a blue bar indicates 'Charges de travail'. The left sidebar contains a navigation menu with categories like 'Workloads', 'Service', 'Config and Storage', and 'Cluster'. The main content area displays three large green circles representing the status of 'Déploiements', 'Pods', and 'Replica Sets', each with a 'Running' indicator. Below this, a table titled 'Déploiements' lists various deployment objects with columns for 'Nom', 'Images', 'Étiquettes', 'Pods', and 'Date de création'.

Nom	Images	Étiquettes	Pods	Date de création ↑
gateway-deployment	elmachkourimane/gateway:latest	-	1 / 1	48 minutes ago
complaints-deployment	elmachkourimane/complaints:latest	-	1 / 1	6 hours ago
authregister-deployment	elmachkourimane/authregister:latest	-	1 / 1	7 hours ago
addcontracy-showlist-deployment	elmachkourimane/addcontracy_showlist:latest	-	1 / 1	7 hours ago
angular-deployment	elmachkourimane/angular:1	-	1 / 1	10 hours ago
internship-model-deployment	essalihisara111/recommendation:internship_ml	-	1 / 1	11 hours ago
car-ml-deployment	essalihisara111/recommendation:car_ml	-	1 / 1	13 hours ago
travel-model-deployment	essalihisara111/recommendation:travel_ml	-	1 / 1	14 hours ago

1. **Pod** : La plus petite unité déployable dans Kubernetes, représentant un ou plusieurs conteneurs qui partagent un réseau et un stockage.
2. **Service** : Un objet Kubernetes qui définit un ensemble logique de Pods et une politique d'accès réseau à ces Pods. Il permet d'exposer une application en interne ou en externe.
3. **Deployment** : Un contrôleur Kubernetes qui gère le déploiement d'applications, déclarant l'état souhaité et garantissant que le nombre spécifié de Pods est toujours en cours d'exécution.
4. **Minikube** : Une distribution légère de Kubernetes qui permet de créer un cluster Kubernetes local à des fins de développement et de test. Il est particulièrement utile pour reproduire des environnements de production en local.

Architecture production :



1. Le processus du developpement jusqu'au deploiement :

- a) **Développement du Microservice par le Développeur :** Dans cette phase, le développeur s'attelle à la conception et à l'implémentation du microservice. Cela englobe la création méticuleuse des interfaces de programmation d'applications (API) conformément aux exigences fonctionnelles spécifiques de l'application. Le processus de développement est orienté vers la réalisation de composants logiciels autonomes et spécialisés, répondant ainsi de manière précise aux besoins fonctionnels définis par le cahier des charges.
- b) **Création du Pipeline :** La création d'un pipeline s'inscrit dans la démarche d'automatisation du processus de développement et de déploiement. Dans le contexte de microservices, un pipeline distinct est élaboré dans Jenkins pour chaque microservice. Cette initiative vise à établir une structure automatisée et cohérente permettant la compilation, les tests, la construction des artefacts, et finalement, le déploiement du microservice associé. L'implémentation d'un pipeline individualisé assure une gestion détaillée de chaque composant, favorisant ainsi une intégration continue et une livraison continue tout au long du cycle de vie du microservice.

```

pipeline {
  agent { label 'kupepod' }
  tools {
    maven '3.9.6'
  }
  stages {
    stage('Checkout') {
      steps {
        checkout scmGit(branches: [[name: '*/main']], extensions: [], userRemoteConfigs: [[url: 'https://github.com/Insurance-Recommendation-Project/gateway-service']])
      }
    }
    stage('Build Maven') {
      steps {
        sh 'mvn clean install'
      }
    }
    stage('Build Docker Image') {
      steps {
        script {
          sh 'docker build -t elmachkouriimane/gateway .'
        }
      }
    }
    stage('Push Image') {
      steps {
        script {
          withCredentials([string(credentialsId: 'docker', variable: 'Docker')]) {
            sh "docker login -u elmachkouriimane -p ${Docker}"
            sh "docker push elmachkouriimane/gateway"
          }
        }
      }
    }
    stage('Deploy App') {
      steps {
        script {
          withKubeConfig(caCertificate: '', clusterName: '', contextName: '', credentialsId: 'kubeconfig', namespace: '', restrictKubeConfigAccess: false, serverUrl: '') {
            sh "kubectl apply -f deployment.yaml"
          }
        }
      }
    }
  }
}

```

Le script que vous voyez ci-dessous représente un pipeline Jenkins défini en tant que code (Jenkinsfile). Ce pipeline détaille les étapes nécessaires à la construction, à la création d'une image Docker, au stockage de cette image, et au déploiement ultérieur de l'application. Il est constitué par les étapes suivantes :

1. Étape d'Agent :

- *Description* : Cette étape spécifie l'agent d'exécution du pipeline, identifié par l'étiquette 'kupepod'. Cela indique que le pipeline sera exécuté sur un agent disposant de cette étiquette. Cet agent représente un pod dans le cluster Kubernetes.
- *Objectif* : Assigner un agent spécifique pour l'exécution du pipeline.

2. Étape d'Outils (Tools) :

- *Description* : Cette étape définit les outils nécessaires au pipeline, en spécifiant ici Maven version 3.9.6.
- *Objectif* : Assurer que l'environnement d'exécution du pipeline dispose des outils requis, ici Maven.

3. Étape de Checkout (Vérification) :

- *Description* : Cette étape récupère le code source du référentiel Git spécifié sur la branche 'main'.
- *Objectif* : Récupérer le code source pour permettre la compilation et la construction ultérieures.

4. Étape de Construction Maven :

- *Description* : Cette étape exécute la commande Maven 'clean install' pour construire le projet.
- *Objectif* : Compiler le code source Java et créer un artefact exécutable.

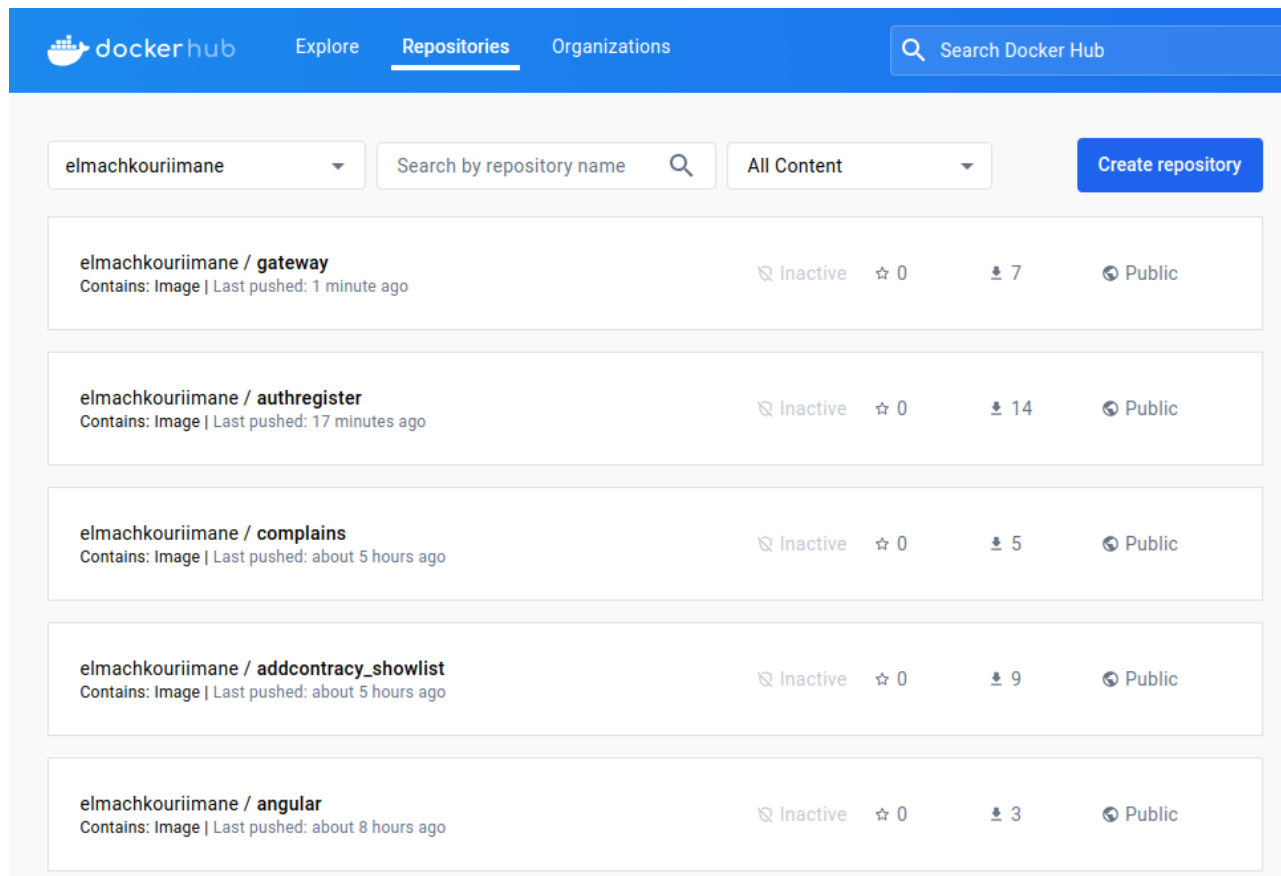
5. Étape de Construction de l'Image Docker :

- *Description* : Cette étape crée une image Docker à partir du code source en utilisant la commande 'docker build' en utilisant un dockerfile .

```
FROM openjdk:21
ADD target/gateway.jar gateway.jar
EXPOSE 8080
ENTRYPOINT ["java", "-jar", "gateway.jar"]
```

- *Objectif* : Générer une image Docker prête à être déployée.

6. Étape de Poussement de l'Image :



- *Description* : Cette étape authentifie l'utilisateur à l'aide des identifiants Docker, puis pousse l'image Docker créée vers un registre Docker.
- *Objectif* : Stocker l'image Docker dans un référentiel partagé.

7. Étape de Déploiement de l'Application :

- *Description* : Cette étape déploie l'application en utilisant Kubernetes. Le fichier 'deployment.yaml' spécifie la configuration du déploiement.
- *Objectif* : Appliquer la configuration du déploiement à l'aide de la commande 'kubectl apply'.

```

apiVersion: apps/v1
kind: Deployment
metadata:
  name: gateway-deployment
spec:
  selector:
    matchLabels:
      app: gateway
  replicas: 1
  template:
    metadata:
      labels:
        app: gateway
    spec:
      containers:
        - name: gateway
          image: elmachkouriimane/gateway:latest
          ports:
            - containerPort: 8080
---
apiVersion: v1
kind: Service
metadata:
  labels:
    app: gateway
  name: gateway-svc
spec:
  selector:
    app: gateway
  type: NodePort
  ports:
    - protocol: TCP
      port: 8080
      targetPort: 8080
status:
  loadBalancer: {}

```

Ce fichier est constitué par deux parties, partie service et partie deployment :

partie deployment:

dedie pour spécifie les paramètres et la configuration nécessaires pour déployer et gérer une application dans un cluster Kubernetes. Il contient généralement les informations suivantes :

1. **apiVersion et kind** : Ces champs définissent la version de l'API Kubernetes utilisée et le type d'objet à créer, tels que "Deployment", "Service", etc.
2. **metadata** : Ce champ contient des métadonnées associées à l'objet, telles que le nom et éventuellement des libellés (labels) pour l'identification.

3. **spec** : C'est la section principale qui spécifie les caractéristiques du déploiement ou du service. Pour un déploiement, cela inclut souvent le nombre de répliques, le sélecteur pour les pods, et le template pour la création des pods.
4. **template** : Présent dans la section spec pour un déploiement, il définit le modèle pour la création des pods. Il contient les spécifications des conteneurs à exécuter, les volumes, et d'autres configurations.
5. **containers** : Dans la section template, la liste des conteneurs à exécuter dans chaque pod est spécifiée. Chaque conteneur a des paramètres tels que le nom, l'image Docker à utiliser, les ports exposés, les variables d'environnement, etc.
6. **ports** : Cela spécifie les ports à exposer pour le service, généralement utilisé dans la configuration d'un service Kubernetes.
7. **selector** : Dans la section spec pour un service, il définit le sélecteur pour trouver les pods associés au service.
8. **type** : Dans la configuration d'un service, il définit le type d'exposition du service, comme ClusterIP, NodePort, ou LoadBalancer.
9. **status** : Il peut contenir des informations sur l'état actuel de l'objet, telles que les adresses IP assignées, les répliques en cours d'exécution, etc.

partie service :

cette partie dédiée pour la spécification des paramètres nécessaires pour exposer et rendre accessible une application au sein du cluster. Voici les éléments généralement présents dans un fichier de configuration de Service Kubernetes :

1. **apiVersion et kind** : Ces champs définissent la version de l'API Kubernetes utilisée et le type d'objet à créer, en l'occurrence, "Service".
2. **metadata** : Cette section contient des métadonnées associées au service, telles que le nom et éventuellement des libellés (labels) pour l'identification.
3. **spec** : C'est la section principale qui spécifie les caractéristiques du service, comprenant notamment :
 - **selector** : Un ensemble de libellés qui identifient les pods auxquels le service doit router le trafic. Ces libellés doivent correspondre aux libellés spécifiés dans les pods associés.
 - **ports** : La liste des ports à exposer sur le service, avec des détails tels que le protocole (TCP ou UDP), le port sur le service, et le port cible sur les pods.
 - **type** : Le type d'exposition du service, qui peut être ClusterIP (accès uniquement à l'intérieur du cluster), NodePort (exposition du service sur chaque nœud du cluster), ou LoadBalancer (utilisation d'un équilibreur de charge externe).
 - **externalIPs** : Une liste d'adresses IP externes facultatives qui peuvent être assignées au service.

- **loadBalancerIP** : L'adresse IP externe facultative à utiliser si le type de service est LoadBalancer.
 - **sessionAffinity** : La configuration pour la session affinity, qui spécifie si le service doit maintenir la persistance des sessions utilisateur.
4. **status** : Il peut contenir des informations sur l'état actuel du service, telles que les adresses IP assignées.

6. Deployment des microservices :

Dans le cadre de l'architecture de l'application, le déploiement des composants est orchestré conformément aux principes de communication et d'exposition des services. En particulier, l'application frontend basée sur Angular est déployée en tant que service de type NodePort, permettant son exposition externe. De manière similaire, le service gateway suit également une configuration NodePort pour faciliter l'accès depuis l'extérieur du cluster. En revanche, les microservices sont configurés en tant que services de type ClusterIP, ce choix stratégique étant motivé par la nécessité de restreindre leurs communications au sein du cluster.

Ce scénario d'exposition différenciée vise à créer une frontière claire entre les composants destinés à être accessibles depuis l'extérieur du cluster, tels que l'application frontend et le service gateway, et ceux destinés à des interactions internes, caractéristiques des microservices. Cette approche garantit une sécurité et une modularité accrues dans le contexte des communications au sein de l'architecture globale de l'application.

Pour le déploiement des microservices, une pratique uniforme a été adoptée pour l'environnement de développement, où tous les microservices, à l'exception de celui dédié aux opérations de machine learning, sont déployés dans un environnement Java. L'utilitaire Maven est utilisé comme outil de construction et de compilation standard pour ces microservices.

7. AngularApp :

L'application frontend de notre système est développée en utilisant le framework Angular, qui, au cours du développement, s'appuie sur son propre serveur de développement intégré. Toutefois, lors du déploiement de l'application, nous avons opté pour l'utilisation du serveur Nginx pour plusieurs raisons stratégiques et opérationnelles :

1. **Performance et Gestion des Requêtes** : Nginx est reconnu pour son efficacité en termes de gestion des requêtes HTTP. Son modèle de traitement asynchrone et non bloquant permet de gérer efficacement un grand nombre de connexions simultanées, améliorant ainsi les performances globales de l'application.
2. **Gestion de la Charge** : Nginx excelle dans la distribution de la charge (load balancing) et la gestion du trafic. Cela permet de répartir équitablement les demandes entre plusieurs instances de l'application frontend, assurant ainsi une utilisation efficace des ressources et une meilleure résilience.
3. **Serveur Web Etabli** : Nginx est largement utilisé comme serveur web dans l'industrie, avec une réputation de fiabilité et de stabilité. Son déploiement généralisé dans des environnements de production en fait un choix éprouvé pour héberger des applications web à grande échelle.
4. **Prise en Charge de Fonctionnalités Avancées** : Nginx offre une variété de fonctionnalités avancées telles que la compression de contenu, la gestion des connexions SSL/TLS, la mise en cache, et la configuration fine des règles de routage. Ces fonctionnalités contribuent à l'optimisation des performances et à l'amélioration de la sécurité.
5. **Réversibilité** : Nginx peut également être configuré comme serveur proxy inversé, permettant de gérer le trafic entrant vers l'application frontend. Cette flexibilité offre des options de configuration avancées pour répondre aux besoins spécifiques de l'architecture déployée.

8. Déploiement du MachineLearning :

Pour le développement de nos applications dédiées au machine learning, nous avons choisi le célèbre framework Python Flask en raison de sa simplicité, de sa flexibilité et de sa robustesse. Cependant, lors du déploiement dans notre environnement de production, nous avons opté pour l'utilisation de Gunicorn en tant que serveur d'exécution pour les raisons suivantes :

- **Performance** : Gunicorn est reconnu pour ses performances élevées et sa capacité à gérer simultanément un grand nombre de requêtes. Son modèle asynchrone garantit une réactivité accrue, ce qui est essentiel pour des applications de machine learning exigeantes.
- **Stabilité et Fiabilité** : Gunicorn est un serveur WSGI (Web Server Gateway Interface) robuste et fiable, largement utilisé dans l'industrie. Il offre une stabilité éprouvée pour le déploiement d'applications Flask en production.

- **Gestion des Requêtes HTTP :** Gunicorn excelle dans la gestion des requêtes HTTP, offrant une couche intermédiaire entre l'application Flask et le serveur web. Cela permet une distribution équilibrée du trafic et une meilleure gestion des connexions.
- **Adaptabilité :** Gunicorn peut être configuré pour fonctionner avec différents serveurs web en amont, offrant une flexibilité accrue. Cette caractéristique permet de s'adapter aux exigences spécifiques de notre environnement de déploiement.
- **Compatibilité avec Kubernetes :** Gunicorn s'intègre bien dans les environnements de déploiement basés sur des conteneurs et peut être facilement orchestré avec des outils tels que Kubernetes, facilitant ainsi la gestion et l'évolutivité des applications Flask.

8. La base de donne :

L'infrastructure de notre application repose sur une interaction étroite avec une base de données MySQL déployée dans le même environnement Kubernetes. Ce déploiement est orchestré à travers une configuration soigneusement définie , mettant en œuvre des concepts tels que Persistent Volume Claims (PVC), Deployments, Services, ConfigMaps, et Secrets.

PersistentVolumeClaim (PVC) :

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: mysql-pv-claim # name of PVC essential for identifying the storage data
  labels:
    app: mysql
    tier: database
spec:
  accessModes:
    - ReadWriteOnce #This specifies the mode of the claim that we are trying to create.
  resources:
    requests:
      storage: 1Gi #This will tell kubernetes about the amount of space we are trying to claim.
```

- **Description :** Le fichier mysql-pv-claim.yaml définit un PersistentVolumeClaim nommé mysql-pv-claim. Il spécifie les besoins en termes de stockage pour le serveur MySQL.
- **Utilisation :** Le PVC est utilisé pour réclamer un volume de stockage persistant d'au moins 1 Gi, assurant que les données du serveur MySQL peuvent persister à travers les redémarrages et les mises à jour.

Deployment :

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: mysql
  labels:
    app: mysql
    tier: database
spec:
  selector: # mysql Pod Should contain same labels
    matchLabels:
      app: mysql
      tier: database
  strategy:
    type: Recreate
  template:
    metadata:
      labels: # Must match 'Service' and 'Deployment' selectors
        app: mysql
        tier: database
    spec:
      containers:
        - image: mysql:5.7 # image from docker-hub
          args:
            - "--ignore-db-dir=lost+found" # Workaround for https://github.com/docker-library/mysql/issues/186
          name: mysql
          env:
            - name: MYSQL_ROOT_PASSWORD
              valueFrom :
                secretKeyRef :
                  name : mysql-secrets
                  key : password
            - name: MYSQL_DATABASE # Setting Database Name from a 'ConfigMap'
              valueFrom :
                configMapKeyRef :
                  name : db-config
                  key : dbName
          ports:
            - containerPort: 3306
              name: mysql
          volumeMounts:
            # Mounting volume obtained from Persistent Volume Claim
            - name: mysql-persistent-storage
              mountPath: /var/lib/mysql #This is the path in the container on which the mounting will take place.
```

- **Description :** Le fichier mysql-deployment.yaml spécifie le déploiement du serveur MySQL en utilisant un Deployment. Il utilise l'image MySQL 5.7, configure les variables d'environnement pour le mot de passe root et le nom de la base de données, et définit le montage du volume persistant.
- **Utilisation :** Le Deployment garantit que le serveur MySQL fonctionne en tant que Pod dans le cluster. Il assure également la gestion de la mise à l'échelle, de la mise à jour et du redémarrage en cas de défaillance.

Service :

```
apiVersion: v1
kind: Service
metadata:
  name: mysql # DNS name
  labels:
    app: mysql
    tier: database
spec:
  ports:
    - port: 3306
      targetPort: 3306
  selector: # mysql Pod Should contain same labels
    app: mysql
    tier: database
  clusterIP: None
```

- **Description :** Le fichier mysql-service.yaml crée un Service nommé mysql exposant le port 3306. Il est configuré en mode clusterIP: None, indiquant que le service ne sera pas accessible au sein du cluster via son IP interne.
- **Utilisation :** Le Service expose le serveur MySQL au sein du cluster, permettant à d'autres services de le joindre pour les opérations de base de données.

ConfigMap :

```
apiVersion : v1
kind : ConfigMap
metadata:
  name : db-config
data:
  host : mysql
  dbName: insuranceDB
```

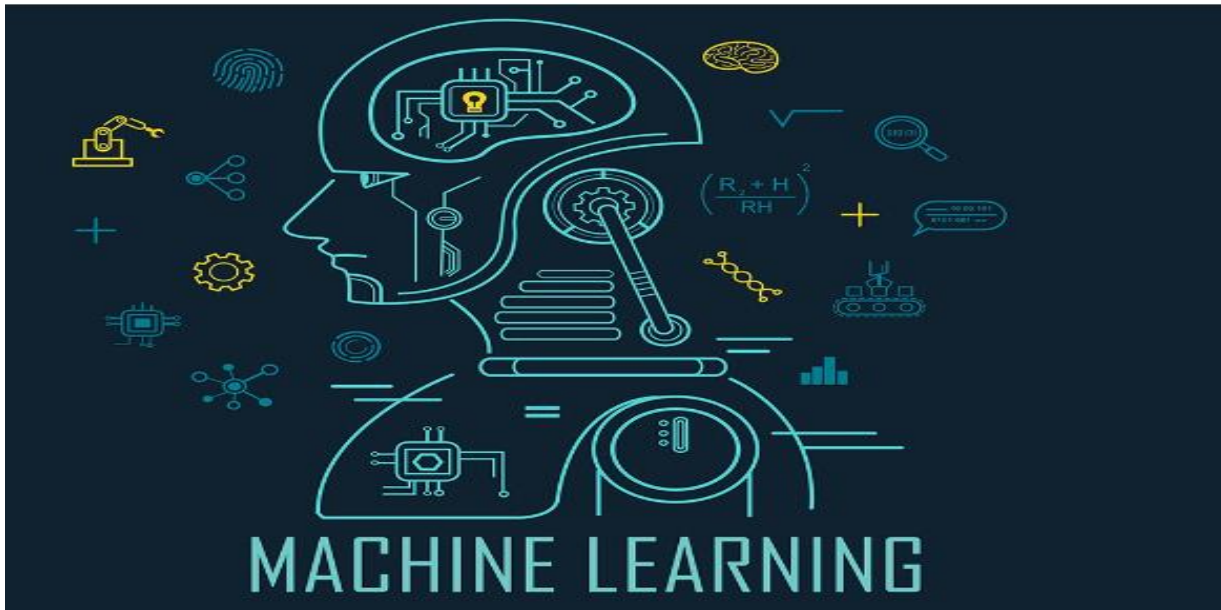
- **Description :** Le fichier db-config.yaml crée un ConfigMap nommé db-config. Il contient les données de configuration, notamment le nom de l'hôte MySQL et le nom de la base de données.
- **Utilisation :** Le ConfigMap est utilisé pour stocker la configuration de l'application, permettant une séparation claire entre la configuration et le déploiement de l'application. Dans ce cas, il est utilisé pour spécifier les détails de la base de données.

Secret :

```
apiVersion : v1
kind : Secret
metadata:
  name : mysql-secrets
data:
  username : cm9vdA==
  password : cm9vdA==
```

- **Description :** Le fichier mysql-secret.yaml définit un Secret nommé mysql-secrets. Il contient les informations sensibles, telles que le nom d'utilisateur et le mot de passe, nécessaires pour accéder au serveur MySQL.
- **Utilisation :** Le Secret est utilisé pour stocker des informations sensibles de manière sécurisée et confidentielle. Dans ce contexte, il s'agit des informations d'authentification pour accéder à la base de données.

III. Machine learning



1. Collecte de Données :

Pour ce projet, la sélection des types d'assurance s'est basée sur une approche diversifiée visant à explorer différents aspects du secteur de l'assurance. Les trois types d'assurance choisis, à savoir **l'assurance automobile**, **l'assurance voyage** et **l'assurance stage**, ont été sélectionnés pour leur pertinence dans des contextes variés. Les ensembles de données correspondants ont été exportés depuis **Kaggle**, une plateforme bien établie pour le partage de données en sciences des données et en apprentissage automatique, garantissant ainsi une diversité et une qualité de données adéquates pour l'entraînement des modèles.

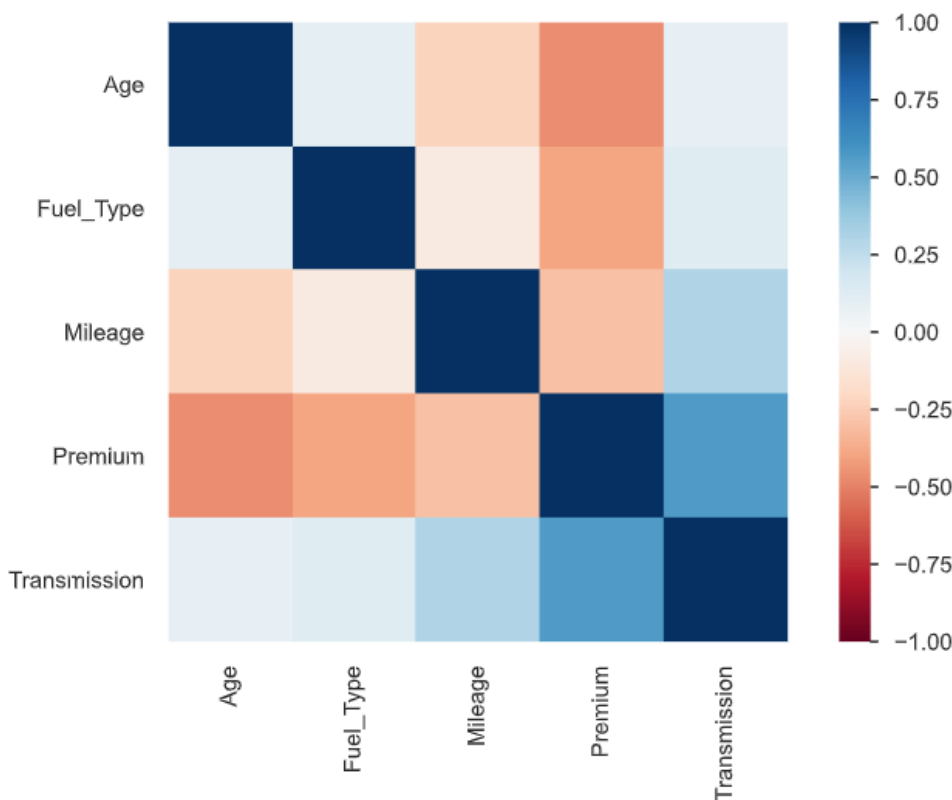
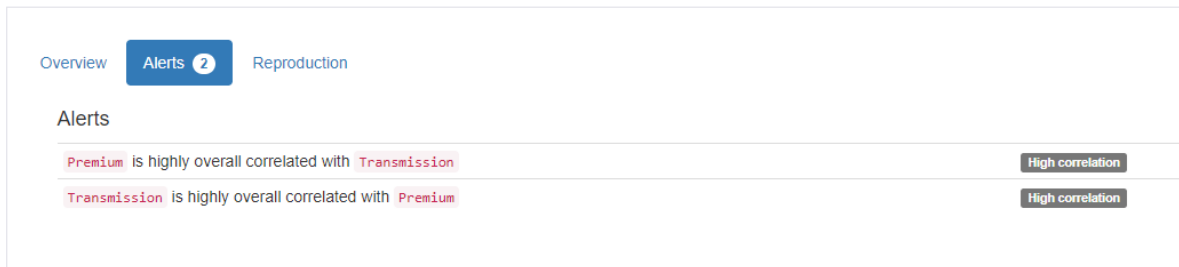
2. Prétraitement de Données :

Dans la phase de prétraitement des données, une attention particulière a été accordée à garantir la qualité et la cohérence des ensembles de données. Les méthodes de traitement des **valeurs nulles** ont été appliquées de manière rigoureuse pour minimiser l'impact des données manquantes sur les modèles. Les doublons ont été identifiés et éliminés afin d'éviter toute redondance dans les jeux de données. Pour traiter les **valeurs aberrantes**, la méthode de l'**IQR** a été employée, permettant une détection robuste des points atypiques. Cette approche statistique a permis de maintenir l'intégrité des données tout en réduisant l'influence des valeurs extrêmes sur les modèles.

En outre, l'utilisation de **diagrammes de corrélation** a joué un rôle essentiel dans l'exploration des relations entre les variables. Ces diagrammes ont facilité la compréhension des interconnexions au sein des ensembles de données, aidant ainsi à prendre des décisions éclairées lors du prétraitement des données. Ces méthodes ont contribué à garantir la fiabilité des données et à créer des bases solides pour l'entraînement des modèles de machine learning.

Exemple de corrélation :

Overview



3. Reduction du Dimension (t-SNE, PCA) :

La réduction de dimensionnalité joue un rôle crucial dans l'optimisation de l'efficacité de nos modèles en transformant des données de grande dimension en un espace de plus basse dimension tout en préservant les motifs essentiels. Dans le cadre de ce projet, deux techniques distinctes, t-SNE et PCA, ont été stratégiquement appliquées à des ensembles de données spécifiques.

Pour les ensembles de données '**Voyage**' et '**Stage**', t-SNE a été utilisé comme technique de réduction de dimensionnalité **non linéaire**. t-SNE excelle dans la capture de relations complexes entre les points de données, surtout dans des scénarios où des structures non linéaires sont prédominantes. En utilisant t-SNE, nous avons cherché à révéler des motifs et des clusters cachés au sein des ensembles de données 'Voyage' et 'Stage', facilitant une compréhension plus nuancée des données sous-jacentes.

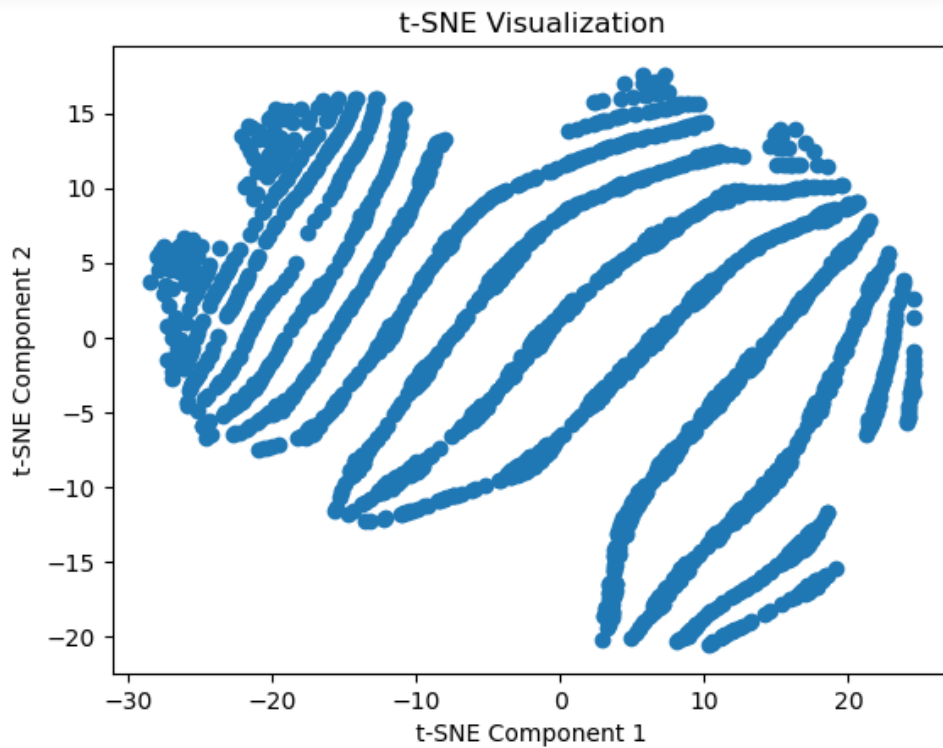
D'autre part, l'ensemble de données '**Voiture**' a subi une réduction de dimensionnalité à l'aide de **PCA**, une technique linéaire largement utilisée pour capturer les composantes principales des données. PCA aide à réduire la redondance et à mettre en évidence les caractéristiques les plus significatives. En appliquant PCA à l'ensemble de données 'Voiture', nous avons cherché à simplifier l'information tout en préservant les caractéristiques essentielles liées à l'assurance automobile.

Ces approches adaptées à la réduction de dimensionnalité sont en accord avec les caractéristiques diverses de chaque ensemble de données, optimisant la capacité des modèles à extraire des insights et des motifs significatifs à partir des données.

Exemple dataset d'assurance de voiture après de PCA :

	PCA1	PCA2
0	0.624538	-0.533864
1	-1.288657	0.628161
2	1.400501	0.890583
3	0.644289	0.104226
4	-0.072177	-0.069484
...
5688	0.085768	-0.347840
5689	0.115542	-0.657272
5690	2.261549	1.242528
5691	-1.372955	-0.025578
5692	-0.237767	0.523669

Exemple de t-SNE :



4. Clustering (EM) :

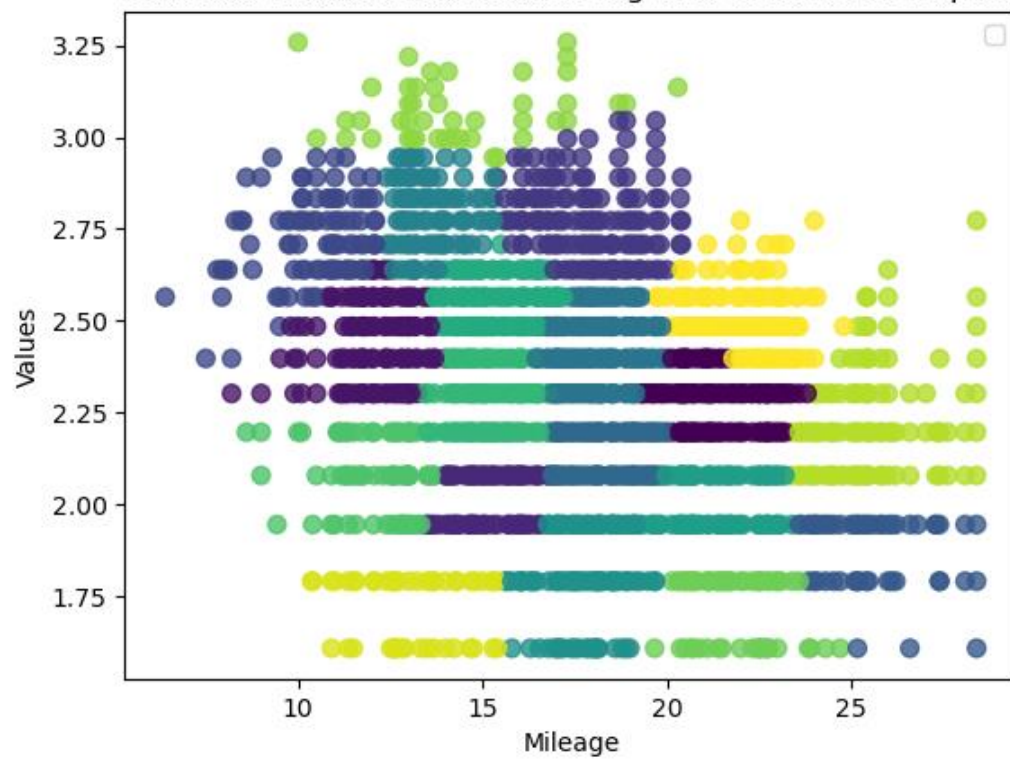
Dans la phase de clustering, l'algorithme **Expectation-Maximization** (EM) a été choisi comme méthode principale pour identifier les structures inhérentes aux données. L'approche EM est particulièrement adaptée pour la modélisation des distributions de probabilité, ce qui en fait un choix judicieux pour le regroupement des données.

L'utilisation de l'algorithme EM a permis d'assigner chaque point de données à un cluster en maximisant la vraisemblance des données observées. Cette approche a facilité la détection de groupes similaires au sein des ensembles de données 'Voyage' et 'Stage', contribuant ainsi à une segmentation précise des données.

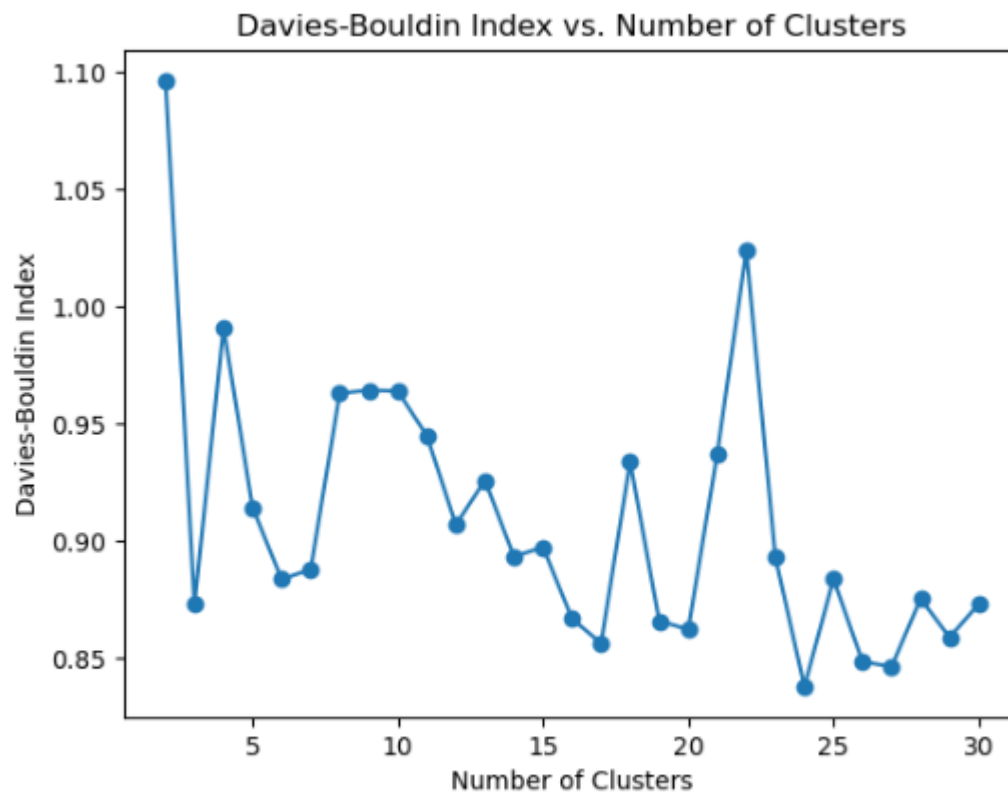
La sélection de l'algorithme EM pour le clustering s'aligne avec la volonté d'adopter des techniques avancées, favorisant une compréhension approfondie des structures sous-jacentes aux données et permettant une exploration plus fine des regroupements naturels présents dans les ensembles de données.

Exemple de EM :

Gaussian Mixture Model Clustering with Confidence Ellipses



5. Le nombre de Clusters K déterminé :



Optimal Number of Clusters: 24

6. Modèle de Classification (RF) :

Dans le cadre de la modélisation, l'algorithme Random Forest (RF) a été sélectionné en raison de ses caractéristiques **robustes** et de sa capacité à traiter des ensembles de données complexes. L'approche de Random Forest, qui agrège plusieurs arbres de décision, a démontré son efficacité dans la gestion de la non-linéarité des données ainsi que dans la réduction du surajustement.

Le choix de Random Forest a été motivé par sa flexibilité à s'adapter à des tâches diverses, allant de la classification à la régression. Les paramètres spécifiques de l'algorithme, tels que le nombre d'arbres et la profondeur maximale, ont été ajustés de manière à optimiser les performances du modèle pour chaque ensemble de données.

L'intégration de Random Forest dans le processus d'entraînement a permis d'exploiter la puissance du vote majoritaire pour améliorer la précision prédictive et la stabilité des modèles, renforçant ainsi la qualité des résultats obtenus.

7. Evaluation des trois modèles :

Pour évaluer les performances des modèles sur les ensembles de données de classification **multiclasses**, une batterie complète de mesures d'évaluation a été utilisée. Les métriques standard telles

que l'**accuracy** , la **précision**, le **rappel**, la **F1-score** et la **matrice de confusion** ont été calculées pour chaque classe, fournissant ainsi un aperçu détaillé des performances du modèle sur l'ensemble des catégories.

La précision représente la proportion de prédictions correctes parmi les instances prédites comme positives, tandis que le rappel mesure la proportion d'instances positives correctement prédites parmi toutes les instances réellement positives. La F1-score, qui est la moyenne harmonique de la précision et du rappel, fournit une mesure équilibrée entre les deux.

En somme, l'évaluation exhaustive des modèles sur des ensembles de données de classification multiclassées a été réalisée en utilisant un ensemble varié de métriques, fournissant ainsi une évaluation holistique de la performance du modèle dans un contexte de classification multiple.

Exemple pour dataset assurance des voitures :

```
---- Inside get_metrics component ----
Classification Report:
      precision    recall  f1-score   support

0           0.94       0.92       0.93         37
1           0.93       1.00       0.96         39
2           1.00       1.00       1.00         24
3           1.00       1.00       1.00         42
4           1.00       1.00       1.00         28
5           0.94       1.00       0.97         16
6           0.97       0.94       0.95         33
7           0.91       1.00       0.95         51
8           1.00       0.93       0.96         28
9           1.00       0.94       0.97         33
10          1.00       0.96       0.98         28
11          1.00       0.97       0.99         37
12          1.00       0.87       0.93         38
13          0.92       1.00       0.96         22
14          0.89       1.00       0.94         16
15          1.00       1.00       1.00         13
16          1.00       1.00       1.00         31
17          1.00       1.00       1.00         14
18          1.00       1.00       1.00         34

accuracy                0.97         564
macro avg              0.97         0.98         0.97         564
weighted avg           0.97         0.97         0.97         564
```

```
Model Metrics: {'accuracy': 0.97, 'precision': 0.97, 'recall': 0.97, 'entropy': 0.09}
```

IV. Blockchain : Gestion d'Assurance et de Réclamations

Introduction

Le projet de blockchain de l'année s'est concentré sur le développement et le déploiement de contrats intelligents dans les domaines de l'assurance et des réclamations. Cette initiative a nécessité l'utilisation de technologies variées, allant de Hardhat à Sepolia, en passant par Etherscan, Alchemy, Web3, et bien d'autres. Ce rapport détaillera les technologies utilisées, les fonctionnalités des contrats intelligents, et fournira une explication approfondie des fichiers de configuration et du script de déploiement.

1. Technologies Utilisées

1. Blockchain et Smart Contracts

- **Ethereum** : La blockchain Ethereum a été choisie comme plateforme principale pour le développement des contrats intelligents, offrant un environnement sécurisé et décentralisé.
- **Solidity** : Le langage de programmation Solidity a été utilisé pour la rédaction des contrats intelligents en raison de sa compatibilité avec la machine virtuelle Ethereum.

2. Environnement de Développement

- **Hardhat** : Le framework Hardhat a été utilisé pour le développement, le test et le déploiement des contrats intelligents. Il offre des fonctionnalités puissantes pour la gestion du cycle de vie des contrats et des scripts associés.

3. Services Blockchain

- **Sepolia** : Le réseau Sepolia a été choisi comme réseau de test pour le développement et le déploiement des contrats intelligents. La variable `Sep_URL` dans le fichier de configuration spécifie l'URL du nœud Sepolia.

4. Sécurité et Vérification

- **Etherscan** :Le service Etherscan a été utilisé pour vérifier et explorer les contrats intelligents déployés sur Ethereum. La clé API Etherscan a été spécifiée dans la configuration Hardhat.

5. Intégration Frontend

- **Web3** :La bibliothèque Web3 a été utilisée pour interagir avec les contrats intelligents depuis l'interface utilisateur frontend.

6. Gestion des Clés

- **MetaMask** : MetaMask a été utilisé comme portefeuille Ethereum, et sa clé privée associée a été stockée de manière sécurisée dans le fichier de configuration.

7. Déploiement

- **Alchemy** :Le service Alchemy a été utilisé pour fournir une infrastructure de nœuds robuste, garantissant une disponibilité constante des contrats intelligents déployés.

2. Contrats Intelligents

1. Contrat d'Assurance (InsuranceContract)

```
contract InsuranceContract {
    struct Insurance {
        address etudiant;
        address society;
        uint256 amount;
        uint256 dateCreation;
        uint256 durationInDays;
        uint256 dateExpiration;
    }

    mapping(address => Insurance[]) public insurances;

    event InsuranceCreated(address indexed etudiant, address indexed society, uint256 amount, uint256 dateCreation, uint256 durationInDays, uint256 dateExpiration);
    event FundTransferredToSociety(address indexed sender, address indexed society, uint256 amount);
}
```

Structure du Contrat

Le contrat d'assurance comprend une structure de données pour représenter les polices d'assurance, stockées dans une carte associative. Chaque police contient des détails tels que l'adresse de l'étudiant, de la société, le montant assuré, et les dates de création et d'expiration.

Fonctionnalités Principales

- Création de l'Assurance : La méthode `createInsurance` permet la création d'une nouvelle police d'assurance avec des paramètres tels que la durée et le montant.
- Liste des Assurances : La méthode `getInsurances` renvoie la liste des polices d'assurance associées à un utilisateur spécifique.
- Transfert à la Société : La méthode `transferToSociety` permet le transfert des fonds à la société d'assurance.

2. Contrat de Réclamation (ReclamationContract)

```
contract ReclamationContract {
    struct Reclamation {
        address etudiant;
        address society;
        uint256 amount;
        uint timestamp;
    }

    mapping(address => Reclamation[]) public reclamations;
    address payable society;

    event ReclamationSubmitted (address indexed user, address indexed society, uint256 amount);
    event FundTransferredToUser(address indexed user, uint256 amount);
}
```

Structure du Contrat

Le contrat de réclamation stocke les réclamations des utilisateurs dans une carte associative, avec des détails tels que l'adresse de l'étudiant, de la société, le montant réclamé et le timestamp.

Fonctionnalités Principales

- Soumission de Réclamation : La méthode `submitReclamation` permet aux utilisateurs de soumettre une réclamation avec des détails tels que le montant réclamé.
- Liste des Réclamations par Utilisateur : La méthode `getReclamationsByUser` renvoie la liste des réclamations associées à un utilisateur spécifique.
- Transfert à l'Utilisateur : La méthode `transferToUser` permet le transfert des fonds réclamés à l'utilisateur.

3. Intégration Blockchain avec Angular

La connexion avec Angular implique trois parties clés : la connexion à MetaMask, la connexion aux contrats intelligents (via leurs adresses et ABI), et la définition de méthodes d'interaction avec ces contrats. Voici une explication détaillée pour chacune de ces parties en utilisant le service `ContractConnectionService` dans Angular.

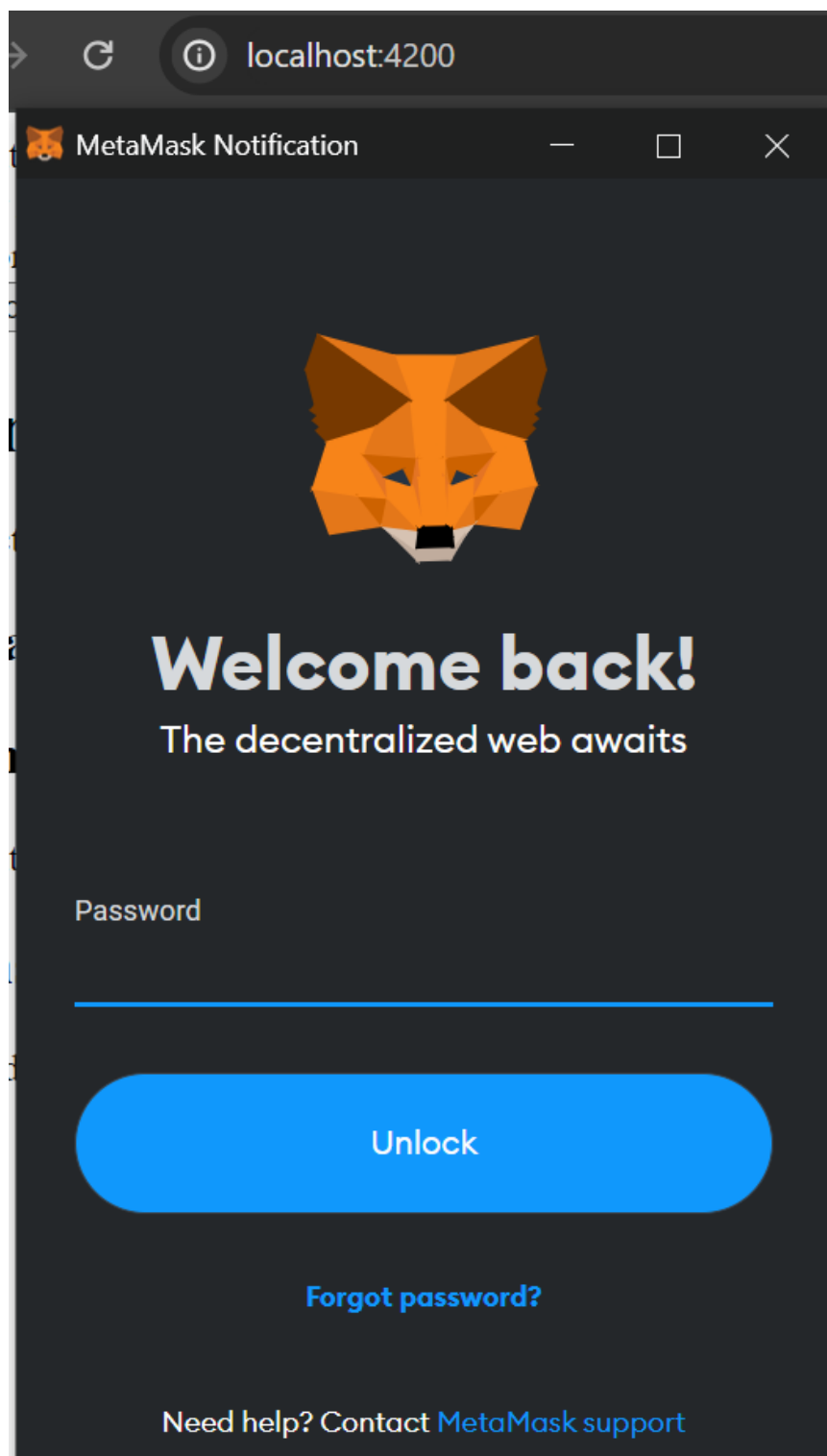
Partie 1: Connexion à MetaMask

```
constructor() {
  console.log('Initializing Web3 Service...');
  if (typeof window !== 'undefined' && window.ethereum) {

    this.connectToMetaMask();
    console.log('MetaMask detected. Initializing Web3...');
    this.web3 = new Web3(window.ethereum);
    this.reclamationContract = new this.web3.eth.Contract(ReclamationContractABI.abi, this.reclamationContractAddress);
    this.insuranceContract = new this.web3.eth.Contract(InsuranceContractABI.abi, this.insuranceContractAddress);
    window.ethereum.on('accountsChanged', (accounts: string[]) => {
      console.log('Accounts changed:', accounts);
      window.location.reload();
    });
  } else {
    console.error('MetaMask not detected!');
  }
}

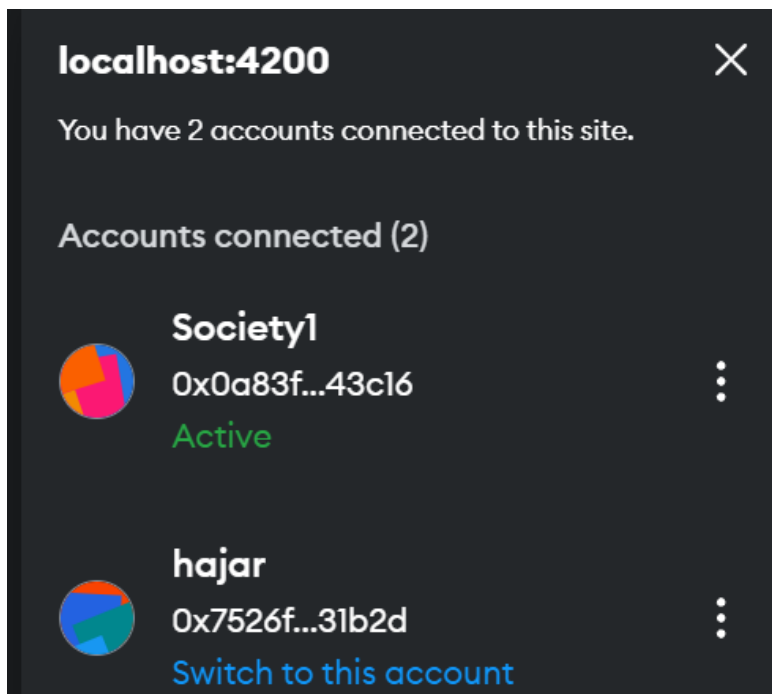
private async connectToMetaMask() {
  try {
    // Request accounts and await user action
    await window.ethereum.request({
      method: 'eth_requestAccounts'
    });

    console.log('Connected to MetaMask!');
  } catch (error) {
    console.error('Error connecting to MetaMask:', error);
  }
}
```



Le service `ContractConnectionService` utilise `window.ethereum` pour détecter la présence de MetaMask. Si MetaMask est détecté, la connexion est établie en utilisant la méthode `connectToMetaMask`. Cette méthode demande l'autorisation à l'utilisateur d'accéder à son compte Ethereum via MetaMask. Une fois connecté, l'objet web3 est initialisé pour interagir avec Ethereum.

Connected Account: 0x7526fb9605FE387929143f626E82125EAf231B2d



Partie 2: Connexion aux Contrats Intelligents

```
this.web3 = new Web3(window.ethereum);  
this.reclamationContract = new this.web3.eth.Contract(ReclamationContractABI.abi, this.reclamationContractAddress);  
this.insuranceContract = new this.web3.eth.Contract(InsuranceContractABI.abi, this.insuranceContractAddress);
```

Les adresses et ABI des contrats intelligents sont définis dans le service `ContractConnectionService`. Lors de l'initialisation, les contrats sont instanciés en utilisant ces adresses et ABI. Cela permet au service d'interagir avec les contrats de manière programmatique.

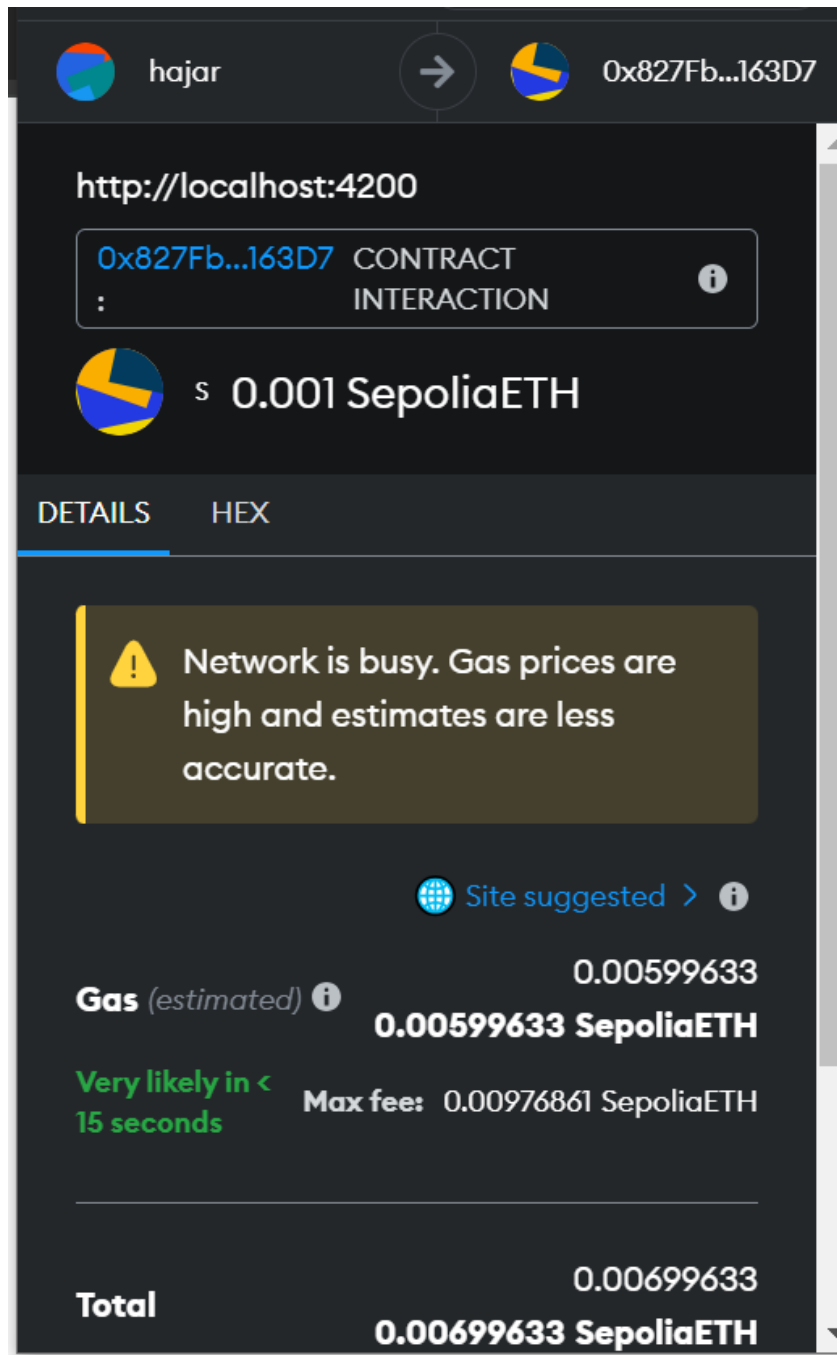
Partie 3: Méthodes d'Interaction avec les Contrats

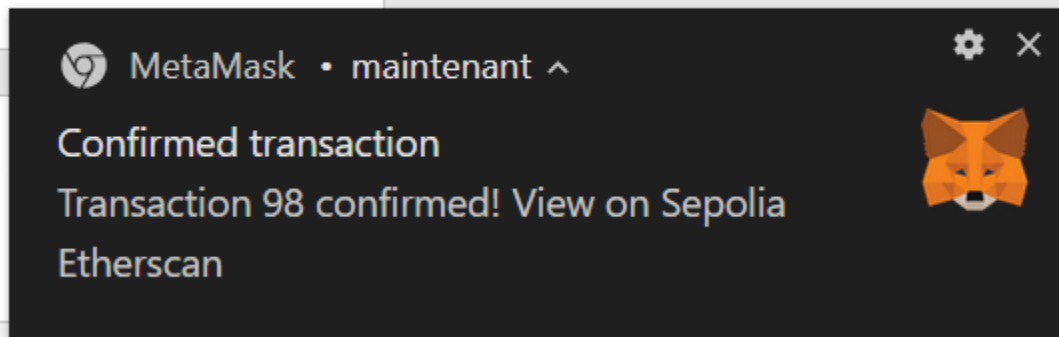
Le service expose plusieurs méthodes permettant à l'interface utilisateur Angular d'interagir avec les contrats intelligents. Ces méthodes incluent :

- `getAccounts` : Récupération des comptes Ethereum associés à MetaMask.
- `submitReclamation` : Soumission d'une réclamation.
- `transferToUser` : Transfert de fonds à un utilisateur.
- `transferToSociety` : Transfert de fonds à la société d'assurance.
- `getReclamationsByUser` : Récupération des réclamations d'un utilisateur.
- `createInsurance` : Création d'une nouvelle police d'assurance.
- `getInsurances` : Récupération des polices d'assurance associées à un utilisateur.

Interaction avec MetaMask

Lors de la connexion à notre application, les utilisateurs sont accueillis par l'interface utilisateur familière de MetaMask. C'est à ce moment que l'application demande l'autorisation d'accéder au compte Ethereum de l'utilisateur. Cette interaction est cruciale, car elle garantit la sécurité et la transparence des transactions effectuées sur la blockchain.





Interaction avec l'Interface Angular

```
constructor(private web3Service: ContractConnectionService, private _snackBar: MatSnackBar) {}

async ngOnInit() {
  try {
    this.accounts = await this.web3Service.getAccounts();

    if (this.accounts.length > 0) {
      this.senderAddress = this.accounts[0];
    } else {
      console.log("no connected");
    }
  } catch (error) {
    console.error('Error in ngOnInit:', error);
  }
}
```

Le service est injecté dans les composants Angular nécessitant une interaction avec les contrats intelligents. Par exemple, une composante Angular pourrait appeler la méthode `submitReclamation` pour permettre à un utilisateur de soumettre une réclamation.

4. Fichiers de Configuration

1. Fichier `hardhat.config.js`

```

require("dotenv").config();
/** @type import('hardhat/config').HardhatUserConfig */

const Sep_URL = process.env.Sep_URL;
const PRIVATE_KEY = process.env.PRIVATE_KEY;
module.exports = {
  solidity: "0.8.17",
  etherscan: {
    apiKey: "7Z9V3BNI8ZIZRRB36WKKGARFR6QKKTN2F6",
  },
  networks: {
    sepolia: {
      url: Sep_URL,
      accounts: [PRIVATE_KEY],
    },
  },
};

```

Le fichier `hardhat.config.js` configure l'environnement de développement et de déploiement. Il spécifie la version de Solidity, les clés API nécessaires, et les paramètres de réseau, notamment l'URL de Sepolia et les clés privées.

Sep_URL

La variable `Sep_URL` représente l'URL du réseau Sepolia, spécifiant le nœud Ethereum à utiliser dans le cadre du développement et du déploiement du projet.

PRIVATE_KEY

La variable `PRIVATE_KEY` stocke la clé privée du compte Ethereum associée au compte MetaMask utilisé pour déployer les contrats intelligents. Garder la clé privée dans une variable d'environnement renforce la sécurité en évitant son exposition directe dans le code source.

etherscanApiKey

La section etherscan dans la configuration Hardhat spécifie la clé API d'Etherscan (`etherscanApiKey`). Cette clé est utilisée pour authentifier les requêtes auprès d'Etherscan, permettant la vérification des contrats déployés.

2. Fichier `deploy.js`

```

const hre = require("hardhat");

async function main() {
  const [deployer] = await hre.ethers.getSigners(); // Obtenez le compte du déploiement

  // Déployez le contrat d'assurance
  const InsuranceContract = await hre.ethers.getContractFactory("InsuranceContract");
  const insurance = await InsuranceContract.deploy();
  await insurance.deployed();
  console.log("Contrat d'assurance déployé à l'adresse:", insurance.address);

  // Déployez le contrat de réclamation
  const ReclamationContract = await hre.ethers.getContractFactory("ReclamationContract");
  const reclamation = await ReclamationContract.deploy();
  await reclamation.deployed();
  console.log("Contrat de réclamation déployé à l'adresse:", reclamation.address);

  // Imprimez l'adresse du déploiement
  console.log("Adresse du déploiement:", deployer.address);
}

main().catch((error) => {
  console.error(error);
  process.exitCode = 1;
});

```

Le fichier `deploy.js` représente le script de déploiement des contrats intelligents. Il utilise Hardhat pour déployer les contrats d'assurance et de réclamation sur la blockchain Sepolia. Les fonctionnalités clés de ce script incluent le déploiement des contrats, l'utilisation de factories de contrats, l'affichage des adresses de déploiement, l'interaction avec l'environnement Hardhat, et la protection des clés privées.

```

PS C:\Users\hp\chainBlockDapp> npx hardhat run --network sepolia scripts/deploy.js
Contrat d'assurance déployé à l'adresse: 0xC7f32EcC519007a68b279757FB7bD27688360B2C
Contrat de réclamation déployé à l'adresse: 0x8d17ADcd8eDd01351E01b279BBf82886b7C9A793
Adresse du déploiement: 0x7526fb9605FE387929143f626E82125EAF231B2d

```

Conclusion

Le projet de blockchain de cette année a été une exploration approfondie de diverses technologies et services liés à Ethereum. De la rédaction des contrats intelligents à leur déploiement sur Sepolia, en passant par l'intégration frontend avec Angular, chaque étape a été soigneusement conçue pour assurer la sécurité, la fonctionnalité et l'expérience utilisateur optimale. Ce rapport vise à fournir une compréhension complète du projet, de ses composants et de ses choix technologiques.

V. Conclusion

En conclusion, notre projet ambitieux d'intégration des smart contrats pour la gestion des contrats d'assurance, associé à un système de recommandation basé sur des algorithmes de machine learning, marque une étape significative dans la transformation numérique du secteur de l'assurance. L'adoption de ces technologies émergentes offre une alternative plus transparente, efficace et économique à la gestion traditionnelle des contrats, tout en fournissant une expérience client plus personnalisée.

La mise en œuvre du backend Spring Boot, combinée à l'utilisation judicieuse des outils DevOps tels que Git, Docker, Kubernetes et Jenkins, garantit la stabilité, la flexibilité et la maintenance aisée de notre application. L'intégration du système de recommandation, déployé à travers Flask et soutenu par des pratiques MLOps, positionne notre solution comme une référence en matière d'innovation dans le secteur de l'assurance.

La gestion du cycle de vie des modèles de machine learning, facilitée par l'utilisation de Kubeflow pour le pipeline, offre une approche holistique et évolutive pour la maintenance et l'amélioration continue de notre système de recommandation. Cette approche MLOps permet une gestion efficace des modèles, assurant ainsi une adaptation continue aux évolutions du marché et aux besoins changeants des clients.

En somme, notre projet n'est pas simplement une avancée technologique, mais une transformation profonde du paysage de l'assurance. Nous aspirons à redéfinir les normes de l'industrie en offrant aux entreprises d'assurance les outils nécessaires pour rester compétitives, tout en offrant aux clients une expérience plus personnalisée et en phase avec leurs besoins spécifiques.

À mesure que notre solution prend forme, elle ouvre la voie à de nouvelles opportunités pour l'industrie de l'assurance, renforçant sa capacité à relever les défis de manière proactive, à anticiper les besoins des clients et à créer des relations durables. En définitive, notre projet s'inscrit dans une démarche de transformation positive du secteur, où l'innovation et la technologie se conjuguent pour façonner l'avenir de l'assurance.

VI. Remerciements

Nous tenons à exprimer notre profonde gratitude envers notre encadrant, le Professeur Lotfi El Achak, dont l'expertise, la guidance et le soutien ont été cruciaux tout au long de notre projet. Son dévouement envers notre réussite a été une source constante d'inspiration et de motivation. Le Professeur El Achak a su partager avec nous sa vaste connaissance du domaine, tout en nous encourageant à explorer de nouvelles perspectives et à repousser les limites de notre compréhension. Ses conseils éclairés ont joué un rôle déterminant dans la direction que notre projet a prise, et nous sommes reconnaissants de pouvoir bénéficier de son expérience. Nous tenons également à souligner la patience et l'engagement constant du Professeur El Achak, qui a pris le temps de discuter, d'analyser et de guider chacune de nos étapes. Sa disponibilité et son enthousiasme ont grandement contribué à faire de notre projet une expérience enrichissante et formatrice. Enfin, nous aimerions exprimer notre reconnaissance pour l'opportunité qui nous a été offerte de travailler sous la supervision du Professeur Lotfi El Achak. Ses enseignements resteront une source d'inspiration continue dans notre parcours académique et professionnel.

Merci sincèrement pour votre dévouement et votre contribution à notre développement académique

Lien github : <https://github.com/Insurance-Recommendation-Project>