TP Intergiciel et programmation par composants

Partie I :

1) La couche **SASL** (Simple Authentication and Security Layer) est une structure fournissant des services d'authentification et de sécurité des données aux protocoles Internet.
2) **SASL** est l'acronyme de Simple Authentication and Security Layer. **SASL** est une couche d'abstraction utilisée dans les protocoles de communication pour ajouter une couche d'authentification et de sécurité.

   SASL permet à un client et à un serveur de négocier et d'établir un mécanisme d'authentification mutuellement acceptable.
3) **SASL** (Simple Authentication and Security Layer) repose sur les principes suivants :
   a) **Abstraction de l'authentification** : SASL fournit une couche d'abstraction pour la gestion de l'authentification. Les applications n'ont pas à gérer directement les détails de l'authentification.
   b) **Flexibilité des mécanismes** : SASL supporte différents mécanismes d'authentification, tels que les mots de passe, les certificats numériques, les jetons d'accès, etc., permettant aux parties de choisir celui qui convient le mieux.
   c) **Négociation** : SASL facilite la négociation entre le client et le serveur pour sélectionner un mécanisme d'authentification mutuellement acceptable.
   d) **Sécurité des données** : SASL offre des services supplémentaires de sécurité tels que l'intégrité des données, la confidentialité et la protection contre la relecture,

e) **Indépendance du protocole** : SASL peut être utilisé dans différents protocoles réseau, assurant une couche d'authentification et de sécurité commune et cohérente entre les différents protocoles.

Partie II :

1) **SASL/GSSAPI (Kerberos) :**

Caractéristiques principales : Utilise le protocole Kerberos pour l'authentification sécurisée basée sur des tickets. Il s'appuie sur une infrastructure d'autorisation centralisée, où les utilisateurs sont authentifiés par un service d'authentification central (AS) et obtiennent un ticket de service pour accéder aux services.

Apport sur la sécurité d'accès à Kafka : Il offre une authentification forte et sécurisée basée sur des tickets Kerberos, garantissant l'identité des utilisateurs et empêchant les accès non autorisés.

2) **SASL/PLAIN :**

Caractéristiques principales : Utilise une authentification simple basée sur des paires d'identifiants (nom d'utilisateur et mot de passe) en texte clair. Il ne fournit pas de mécanisme de chiffrement ou de protection des données d'authentification.

Apport sur la sécurité d'accès à Kafka : Il offre une méthode d'authentification basique mais non sécurisée, ne convenant généralement pas aux environnements de production. Il est principalement utilisé pour des tests ou des développements locaux.

3) **SASL/SCRAM-SHA-256 et SASL/SCRAM-SHA-512 :**

Caractéristiques principales : Utilise le mécanisme SCRAM (Salted Challenge Response Authentication Mechanism) pour l'authentification basée sur des mots de passe sécurisés.

SCRAM-SHA-256 utilise une fonction de hachage SHA-256, tandis que SCRAM-SHA-512 utilise SHA-512.

Apport sur la sécurité d'accès à Kafka : Ils offrent une authentification sécurisée basée sur des mots de passe, empêchant l'utilisation de mots de passe en texte clair. Les mots de passe sont stockés sous forme de hachages sécurisés, garantissant une protection supplémentaire des informations d'identification.

4) **SASL/OAUTHBEARER :**

Caractéristiques principales : Utilise le protocole OAuth pour l'authentification et l'autorisation basées sur des jetons. Il permet l'intégration avec des fournisseurs d'identité externes, tels que Google, Facebook, etc.

Apport sur la sécurité d'accès à Kafka : Il permet l'authentification basée sur des jetons OAuth, offrant une intégration avec des fournisseurs d'identité tiers et une gestion centralisée des autorisations.

En ce qui concerne ZooKeeper, il est recommandé de le sécuriser également. ZooKeeper est utilisé par Kafka pour stocker les métadonnées et les configurations, et il est important de protéger l'accès à ces informations sensibles. On peut sécuriser ZooKeeper en utilisant des mécanismes tels que SASL ou TLS (Transport Layer Security).

ZooKeeper est essentiel pour le fonctionnement de Kafka en tant que registre de métadonnées. Cependant, à partir de la version 2.8 de Kafka, une nouvelle fonctionnalité appelée **Kafka Raft Metadata** Mode est introduite, qui permet à Kafka de fonctionner sans dépendre de ZooKeeper pour le stockage des métadonnées. Cette fonctionnalité utilise un protocole de consensus appelé Raft pour la réplication et la coordination des métadonnées Kafka. Cela signifie que Kafka peut fonctionner sans ZooKeeper dans ce mode.

En ce qui concerne l'utilisation de SASL avec Kafka en mode sans ZooKeeper, Kafka Raft Metadata Mode prend en charge l'authentification SASL en utilisant les mêmes mécanismes SASL mentionnés précédemment.

Partie III :

1. Lancer les dockers containers avec postgres et pgadmin
2. Lancer Zookeeper, le broker, le consumer et le producer.

Mode non sécurisée :

zookeeper.properties

```
# the directory where the snapshot is stored.
dataDir=/tmp/zookeeper
# the port at which the clients will connect
clientPort=2181
# disable the per-ip limit on the number of connections since this is a non-production config
maxClientCnxns=0
# Disable the adminserver by default to avoid port conflicts.
# Set the port to something non-conflicting if choosing to enable this
admin.enableServer=true
admin.serverPort=8080
```

server.properties

```
############################# Server Basics #############################
broker.id=0
############################# Socket Server Settings #############################
listeners=PLAINTEXT://localhost:9092
num.network.threads=3
num.io.threads=8
socket.send.buffer.bytes=102400
socket.receive.buffer.bytes=102400
socket.request.max.bytes=104857600
############################# Log Basics #############################
log.dirs=/tmp/kafka-logs
num.partitions=1
num.recovery.threads.per.data.dir=1
############################# Internal Topic Settings  #############################
offsets.topic.replication.factor=1
transaction.state.log.replication.factor=1
transaction.state.log.min.isr=1
delete.topic.enable=true
############################# Log Retention Policy #############################
log.retention.check.interval.ms=300000
############################# Zookeeper #############################
zookeeper.connect=localhost:2181
zookeeper.connection.timeout.ms=18000
############################# Group Coordinator Settings #############################
group.initial.rebalance.delay.ms=0
```

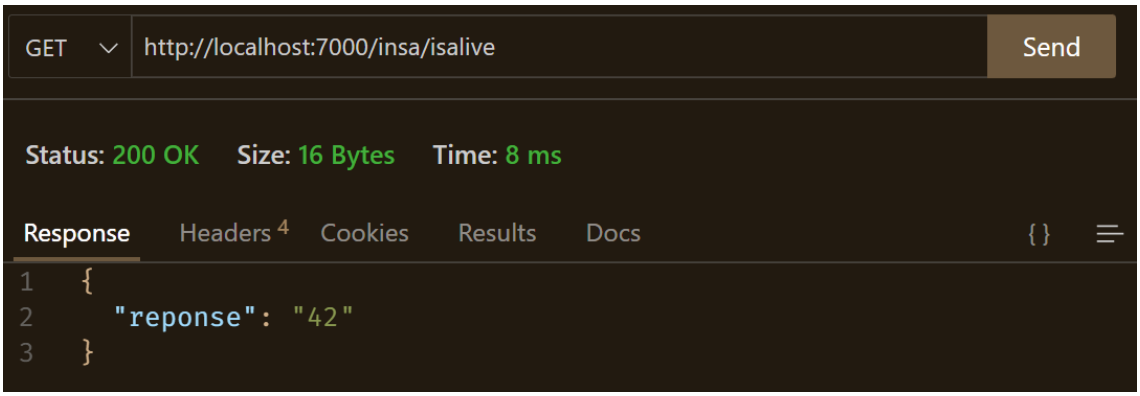
genkp.properties

```
server.port=7000

spring.kafka.producer.bootstrap-servers=localhost:9092
spring.kafka.producer.key-serializer=org.apache.kafka.common.serialization.StringSerializer
spring.kafka.producer.value-serializer=org.apache.kafka.common.serialization.StringSerializer

#definition du nom du topic pour ce service producer
application.topic=ETUDIANTS

```

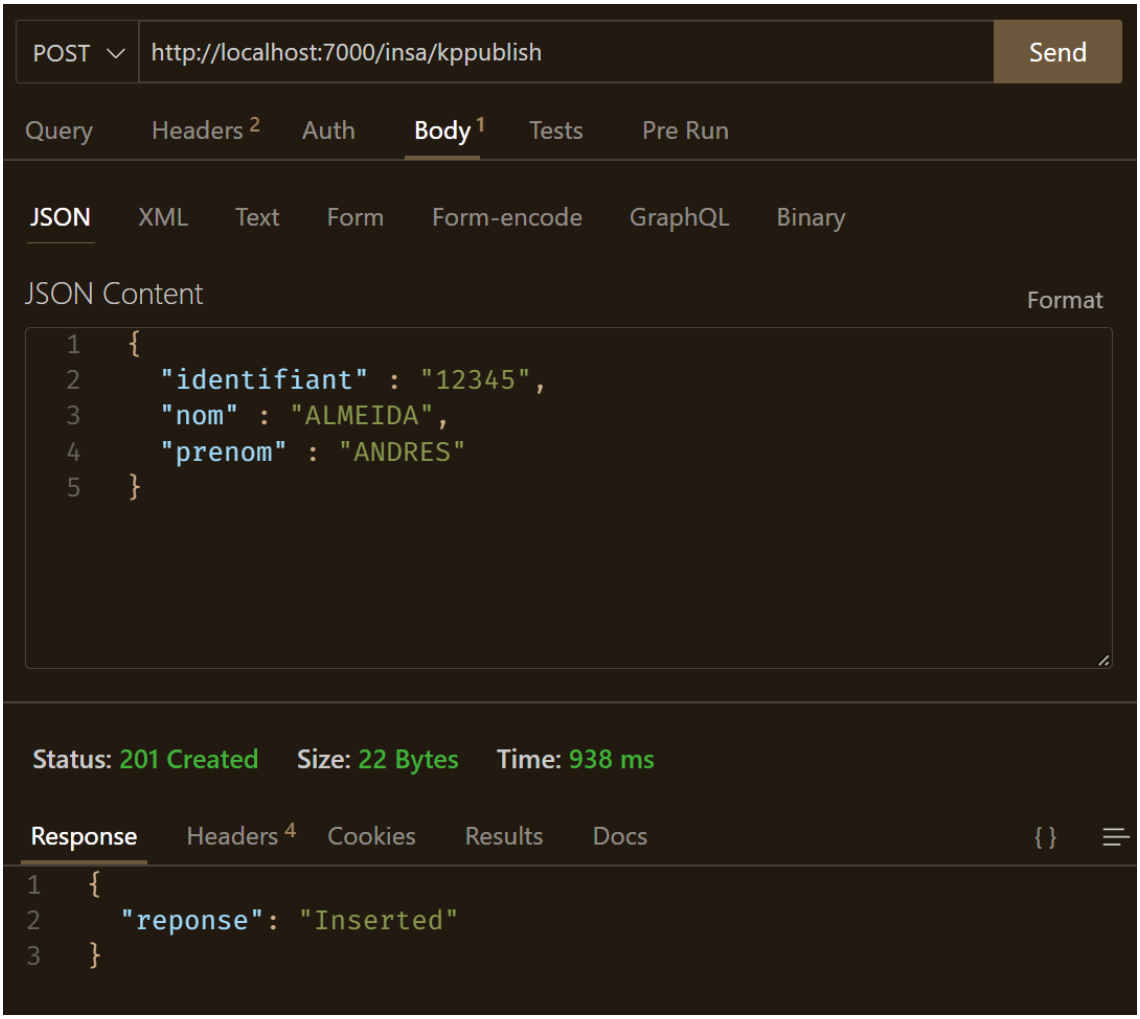
kc-etudiants.properties

```
server.port=7100

spring.kafka.consumer.bootstrap-servers=localhost:9092
spring.kafka.consumer.topic-name=ETUDIANTS
spring.kafka.consumer.group-id=ETUDIANTS_GROUP
spring.kafka.consumer.auto-offset-reset=earliest
spring.kafka.consumer.key-deserializer=org.apache.kafka.common.serialization.StringDeserializer
spring.kafka.consumer.value-deserializer=org.apache.kafka.common.serialization.StringDeserializer

spring.datasource.url=jdbc:postgresql://localhost:5432/etudiants
spring.datasource.username=postgres
spring.datasource.password=admin
spring.datasource.driverClassName=org.postgresql.Driver
```

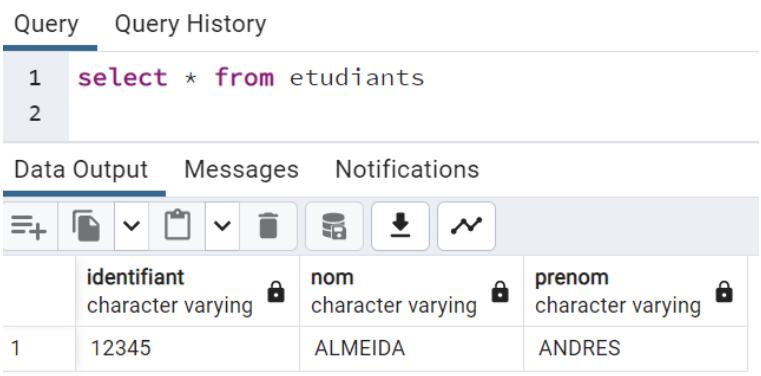
GET request



POST request



Base de données



Mode SASL/PLAIN :

zookeeper.properties

```
# the directory where the snapshot is stored.
dataDir=/tmp/zookeeper
# the port at which the clients will connect
clientPort=2181
# disable the per-ip limit on the number of connections since this is a non-production config
maxClientCnxns=0
# Disable the adminserver by default to avoid port conflicts.
# Set the port to something non-conflicting if choosing to enable this
admin.enableServer=true
admin.serverPort=8080
```

server_sasl_plain.properties

```
############################ Server Basics #############################
broker.id=0
############################ Socket Server Settings ############################
listeners=SASL_PLAINTEXT://localhost:9092
advertised.listeners=SASL_PLAINTEXT://localhost:9092
num.network.threads=3
num.io.threads=8
socket.send.buffer.bytes=102400
socket.receive.buffer.bytes=102400
socket.request.max.bytes=104857600
############################ Log Basics ############################
log.dirs=/tmp/kafka-logs
num.partitions=1
num.recovery.threads.per.data.dir=1
############################ Internal Topic Settings  ############################
offsets.topic.replication.factor=1
transaction.state.log.replication.factor=1
transaction.state.log.min.isr=1
delete.topic.enable=true
############################ Log Retention Policy ############################
log.retention.check.interval.ms=300000
############################ Zookeeper ############################
zookeeper.connect=localhost:2181
zookeeper.connection.timeout.ms=18000
############################ Group Coordinator Settings ############################
group.initial.rebalance.delay.ms=0
############################ SASL ############################
security.inter.broker.protocol=SASL_PLAINTEXT
sasl.enabled.mechanisms=PLAIN
sasl.mechanism.inter.broker.protocol=PLAIN
listener.name.sasl_plaintext.plain.sasl.jaas.config=org.apache.kafka.common.security.plain.PlainLoginModule required \
    username="admin" \
    password="admin-secret" \
    user_admin="admin-secret" \
    user_consumer="consumer-secret" \
    user_producer="producer-secret";
```

La propriété **listeners** spécifie les protocoles et les ports sur lesquels le courtier Kafka écoute. Nous avons ajouté SASL_PLAINTEXT://:9092 pour activer l'authentification SASL sur le port 9092.

Les propriétés **sasl.mechanism.inter.broker.protocol** et **sasl.enabled.mechanisms** spécifient les mécanismes SASL autorisés. Ici, nous utilisons PLAIN comme mécanisme SASL.

On a ajouté la propriété **listener.name.sasl_plaintext.plain.sasl.jaas.config** pour inclure les informations d'authentification des utilisateurs. On utilisera ces utilisateurs pour authentifier le consumer et le producer.

genkp.properties

```
server.port=7000

spring.kafka.producer.bootstrap-servers=localhost:9092
spring.kafka.producer.key-serializer=org.apache.kafka.common.serialization.StringSerializer
spring.kafka.producer.value-serializer=org.apache.kafka.common.serialization.StringSerializer

#definition du nom du topic pour ce service producer
application.topic=ETUDIANTS

spring.kafka.jaas.enabled=true
spring.kafka.properties.security.protocol=SASL_PLAINTEXT
spring.kafka.properties.sasl.mechanism=PLAIN
spring.kafka.properties.sasl.jaas.config=org.apache.kafka.common.security.plain.PlainLoginModule required \
    username="producer" \
    password="producer-secret";
```

kc-etudiants.properties

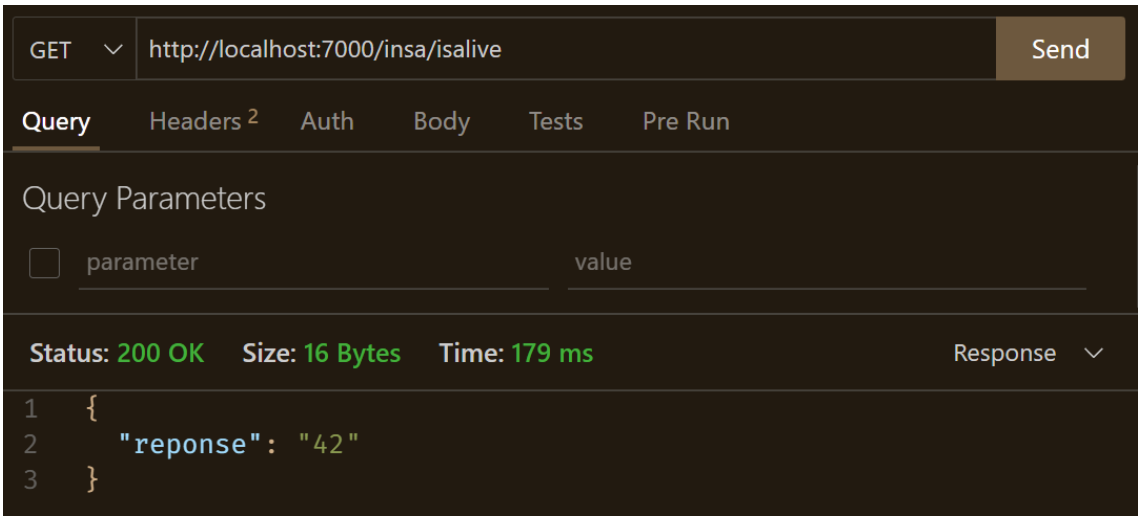
```
server.port=7100

spring.kafka.consumer.bootstrap-servers=localhost:9092
spring.kafka.consumer.topic-name=ETUDIANTS
spring.kafka.consumer.group-id=ETUDIANTS_GROUP
spring.kafka.consumer.auto-offset-reset=earliest
spring.kafka.consumer.key-deserializer=org.apache.kafka.common.serialization.StringDeserializer
spring.kafka.consumer.value-deserializer=org.apache.kafka.common.serialization.StringDeserializer

spring.datasource.url=jdbc:postgresql://localhost:5432/etudiants
spring.datasource.username=postgres
spring.datasource.password=admin
spring.datasource.driverClassName=org.postgresql.Driver

spring.kafka.jaas.enabled=true
spring.kafka.properties.security.protocol=SASL_PLAINTEXT
spring.kafka.properties.sasl.mechanism=PLAIN
spring.kafka.properties.sasl.jaas.config=org.apache.kafka.common.security.plain.PlainLoginModule required \
    username="consumer" \
    password="consumer-secret";
```

La propriété **spring.kafka.producer.properties.security.protocol** définit le protocole de sécurité utilisé par le producteur Kafka. Il est configuré sur "SASL_PLAINTEXT" pour activer l'authentification SASL.

La propriété **spring.kafka.producer.properties.sasl.mechanism** spécifie le mécanisme SASL utilisé par le producteur Kafka. Il est configuré sur “PLAIN” est configuré pour utiliser SASL PLAIN.

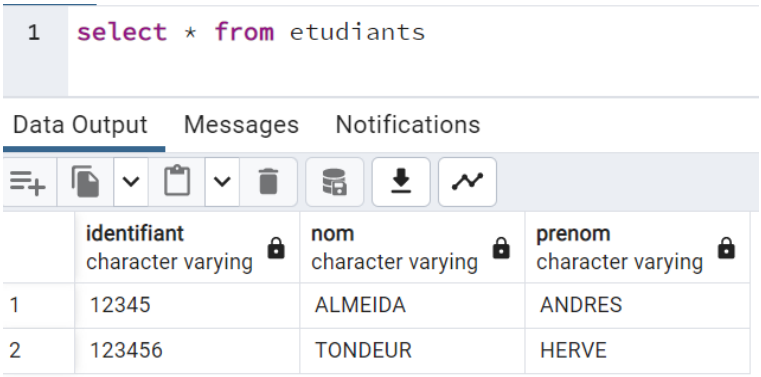
La propriété **spring.kafka.producer.properties.sasl.jaas.config** spécifie la configuration JAAS pour l'authentification SASL du producteur Kafka avec le module JAAS à utiliser.

GET request



POST request

Base de données



```
1  select * from etudiants
```

Data Output | Messages | Notifications

	identifiant character varying	nom character varying	prenom character varying
1	12345	ALMEIDA	ANDRES
2	123456	TONDEUR	HERVE

Comparaison

Lorsque SASL PLAIN est configuré pour l'authentification dans Kafka, on constate que les requêtes GET et POST prennent plus de temps à répondre dans les tests. Cela peut être dû à l'ajout d'un processus d'authentification supplémentaire lors de l'établissement de la connexion avec le courtier Kafka.

Lorsqu'un client envoie une requête à un courtier Kafka, il doit d'abord s'authentifier avec les informations d'identification fournies (nom d'utilisateur et mot de passe) en utilisant le mécanisme SASL PLAIN. Le courtier Kafka vérifie ensuite ces informations d'identification avant de répondre à la requête.