**Exercice 1:**

1. Développez un job Spring Batch qui lit les données depuis un fichier CSV, transforme les informations en calculant la durée de séjour pour chaque patient, puis calcule la durée moyenne des séjours par service, et stocke les résultats dans une base de données en mémoire (par exemple, H2).

package ma.oulakbir.Exercice1;  
  
public record Patient(Long id, String nom, String service, String dateAdmission, String dateSortie) {}

package ma.oulakbir.Exercice1;  
  
import org.springframework.batch.item.ItemProcessor;  
  
import java.time.LocalDate;  
import java.time.temporal.ChronoUnit;  
import java.util.ArrayList;  
import java.util.HashMap;  
import java.util.List;  
import java.util.Map;  
  
public class PatientProcessor implements ItemProcessor<Patient, ServiceSummary> {  
  
 private final Map<String, List<Long>> serviceDurations = new HashMap<>();  
  
 @Override  
 public ServiceSummary process(Patient patient) throws Exception {  
 if (patient.dateSortie() == null || patient.dateSortie().isEmpty()) {  
 return null; // Skip records with missing DateSortie  
 }  
  
 // Calculate stay duration  
 LocalDate admissionDate = LocalDate.*parse*(patient.dateAdmission());  
 LocalDate dischargeDate = LocalDate.*parse*(patient.dateSortie());  
 long duration = ChronoUnit.*DAYS*.between(admissionDate, dischargeDate);  
  
 // Aggregate durations by service  
 serviceDurations.computeIfAbsent(patient.service(), k -> new ArrayList<>()).add(duration);  
  
 // Calculate average duration for the service  
 double avgDuration = serviceDurations.get(patient.service()).stream()  
 .mapToLong(Long::longValue)  
 .average()  
 .orElse(0);  
  
 return new ServiceSummary(patient.service(), avgDuration);

package ma.oulakbir.Exercice1;  
  
import org.springframework.batch.core.Job;  
import org.springframework.batch.core.Step;  
import org.springframework.batch.core.job.builder.JobBuilder;  
import org.springframework.batch.core.repository.JobRepository;  
import org.springframework.batch.core.step.builder.StepBuilder;  
import org.springframework.batch.item.ItemProcessor;  
import org.springframework.batch.item.database.BeanPropertyItemSqlParameterSourceProvider;  
import org.springframework.batch.item.database.JdbcBatchItemWriter;  
import org.springframework.batch.item.database.builder.JdbcBatchItemWriterBuilder;  
import org.springframework.batch.item.file.FlatFileItemReader;  
import org.springframework.batch.item.file.builder.FlatFileItemReaderBuilder;  
import org.springframework.context.annotation.Bean;  
import org.springframework.context.annotation.Configuration;  
import org.springframework.core.io.ClassPathResource;  
import org.springframework.jdbc.datasource.DataSourceTransactionManager;  
  
import javax.sql.DataSource;  
import org.springframework.jdbc.datasource.DriverManagerDataSource;  
  
@Configuration  
public class BatchConfiguration {  
  
 // H2 DataSource configuration  
 @Bean  
 public DataSource dataSource() {  
 DriverManagerDataSource dataSource = new DriverManagerDataSource();  
 dataSource.setDriverClassName("org.h2.Driver");  
 dataSource.setUrl("jdbc:h2:mem:hospital-db;DB\_CLOSE\_DELAY=-1;MODE=PostgreSQL");  
 dataSource.setUsername("sa");  
 dataSource.setPassword("");  
 return dataSource;  
 }  
  
 @Bean  
 public DataSourceTransactionManager transactionManager(DataSource dataSource) {  
 return new DataSourceTransactionManager(dataSource);  
 }  
  
 @Bean  
 public FlatFileItemReader<Patient> reader() {  
 return new FlatFileItemReaderBuilder<Patient>()  
 .name("patientItemReader")  
 .resource(new ClassPathResource("hospitalizations.csv"))  
 .delimited()  
 .names("id", "nom", "service", "dateAdmission", "dateSortie")  
 .linesToSkip(1)  
 .targetType(Patient.class)  
 .build();  
 }  
  
 @Bean  
 public ItemProcessor<Patient, ServiceSummary> processor() {  
 return new PatientProcessor();  
 }  
  
 @Bean  
 public JdbcBatchItemWriter<ServiceSummary> writer(DataSource dataSource) {  
 return new JdbcBatchItemWriterBuilder<ServiceSummary>()  
 .itemSqlParameterSourceProvider(new BeanPropertyItemSqlParameterSourceProvider<>())  
 .sql("MERGE INTO hospitalization\_summary (service, avg\_duration) KEY(service) VALUES (:service, :avgDuration)")  
 .dataSource(dataSource)  
 .build();  
 }  
  
 @Bean  
 public Step step(JobRepository jobRepository, DataSourceTransactionManager transactionManager,  
 FlatFileItemReader<Patient> reader, ItemProcessor<Patient, ServiceSummary> processor,  
 JdbcBatchItemWriter<ServiceSummary> writer) {  
 return new StepBuilder("step1", jobRepository)  
 .<Patient, ServiceSummary>chunk(3, transactionManager)  
 .reader(reader)  
 .processor(processor)  
 .writer(writer)  
 .build();  
 }  
  
 @Bean  
 public Job importPatientJob(JobRepository jobRepository, Step step, JobCompletionNotificationListener listener) {  
 return new JobBuilder("importPatientJob", jobRepository)  
 .listener(listener)  
 .start(step)  
 .build();  
 }  
}

2. Modifiez le job précédent pour ignorer les enregistrements où la date de sortie est absente.

package ma.oulakbir.Exercice1;  
  
import org.springframework.batch.item.ItemProcessor;  
  
import java.time.LocalDate;  
import java.time.temporal.ChronoUnit;  
import java.util.ArrayList;  
import java.util.HashMap;  
import java.util.List;  
import java.util.Map;  
  
public class PatientProcessor implements ItemProcessor<Patient, ServiceSummary> {  
  
 private final Map<String, List<Long>> serviceDurations = new HashMap<>();  
  
 @Override  
 public ServiceSummary process(Patient patient) throws Exception {  
 if (patient.dateSortie() == null || patient.dateSortie().isEmpty()) {  
 return null; // Skip records with missing DateSortie  
 }  
  
 // Calculate stay duration  
 LocalDate admissionDate = LocalDate.*parse*(patient.dateAdmission());  
 LocalDate dischargeDate = LocalDate.*parse*(patient.dateSortie());  
 long duration = ChronoUnit.*DAYS*.between(admissionDate, dischargeDate);  
  
 // Aggregate durations by service  
 serviceDurations.computeIfAbsent(patient.service(), k -> new ArrayList<>()).add(duration);  
  
 // Calculate average duration for the service  
 double avgDuration = serviceDurations.get(patient.service()).stream()  
 .mapToLong(Long::longValue)  
 .average()  
 .orElse(0);  
  
 return new ServiceSummary(patient.service(), avgDuration);  
 }  
}

package ma.oulakbir.Exercice1;  
  
public record ServiceSummary(String service, Double avgDuration) {}

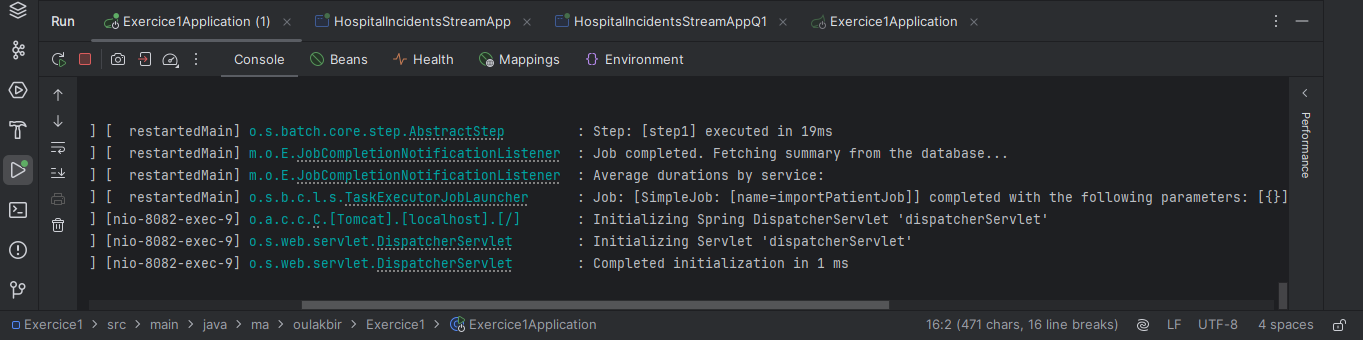
3. Modifiez le premier job Spring Batch pour qu'il soit automatiquement planifié et s'exécute tous les jours à minuit. Ce job doit lire les données depuis un fichier CSV, transformer les informations en calculant la durée de séjour pour chaque patient, calculer la durée moyenne des séjours par service, et stocker les résultats dans une base de données en mémoire (par, exemple, H2).

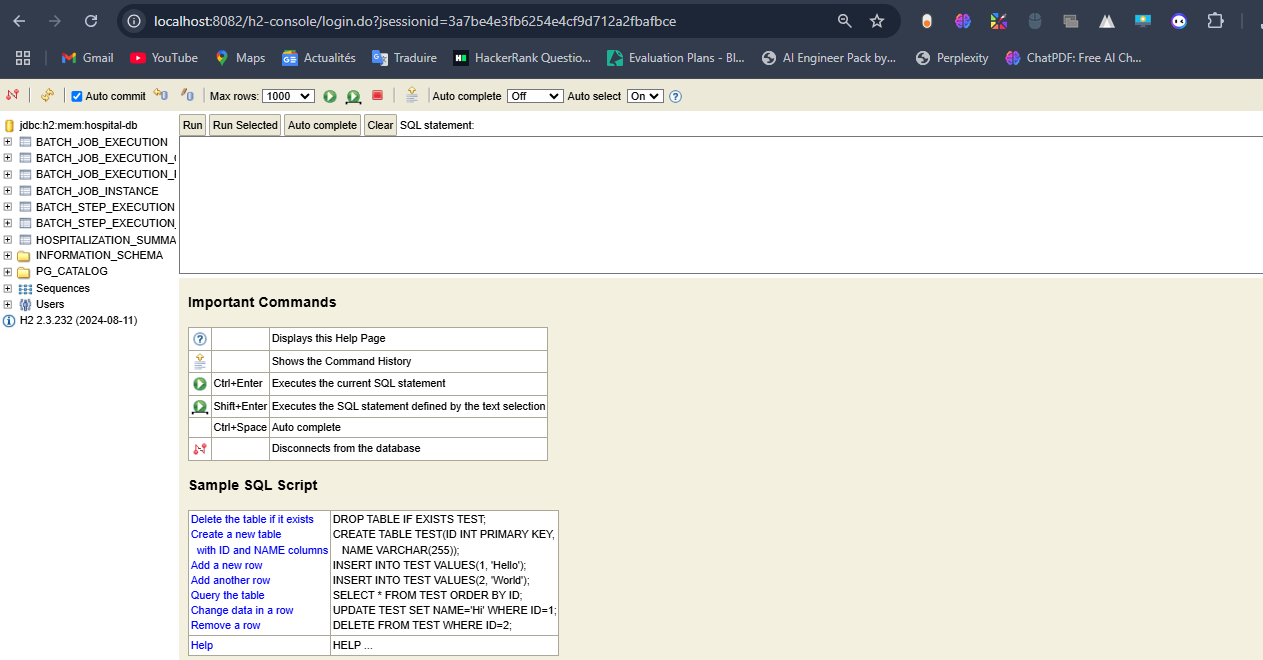
package ma.oulakbir.Exercice1;  
  
import org.slf4j.Logger;  
import org.slf4j.LoggerFactory;  
import org.springframework.batch.core.JobExecution;  
import org.springframework.batch.core.JobExecutionListener;  
import org.springframework.jdbc.core.DataClassRowMapper;  
import org.springframework.jdbc.core.JdbcTemplate;  
import org.springframework.stereotype.Component;  
  
import java.util.List;  
  
@Component  
public class JobCompletionNotificationListener implements JobExecutionListener {  
  
 private static final Logger *logger* = LoggerFactory.*getLogger*(JobCompletionNotificationListener.class);  
  
 private final JdbcTemplate jdbcTemplate;  
  
 public JobCompletionNotificationListener(JdbcTemplate jdbcTemplate) {  
 this.jdbcTemplate = jdbcTemplate;  
 }  
  
 @Override  
 public void afterJob(JobExecution jobExecution) {  
 *logger*.info("Job completed. Fetching summary from the database...");  
  
 List<ServiceSummary> summaries = jdbcTemplate.query(  
 "SELECT service, avg\_duration FROM hospitalization\_summary",  
 new DataClassRowMapper<>(ServiceSummary.class));  
  
 *logger*.info("Average durations by service:");  
 summaries.forEach(summary -> *logger*.info("Service: {}, Average Duration: {}", summary.service(), summary.avgDuration()));  
 }  
}

@Bean  
public FlatFileItemReader<Patient> reader() {  
 return new FlatFileItemReaderBuilder<Patient>()  
 .name("patientItemReader")  
 .resource(new ClassPathResource("hospitalizations.csv"))  
 .delimited()  
 .names("id", "nom", "service", "dateAdmission", "dateSortie")  
 .linesToSkip(1)  
 .targetType(Patient.class)  
 .build();  
}

package ma.oulakbir.Exercice1;  
  
import org.springframework.boot.SpringApplication;  
import org.springframework.boot.autoconfigure.SpringBootApplication;  
import org.springframework.core.io.ClassPathResource;  
import org.springframework.scheduling.annotation.EnableScheduling;  
  
@SpringBootApplication  
@EnableScheduling  
public class Exercice1Application {  
  
 public static void main(String[] args) {  
 SpringApplication.*run*(Exercice1Application.class, args);  
 }  
  
}

Resultats:



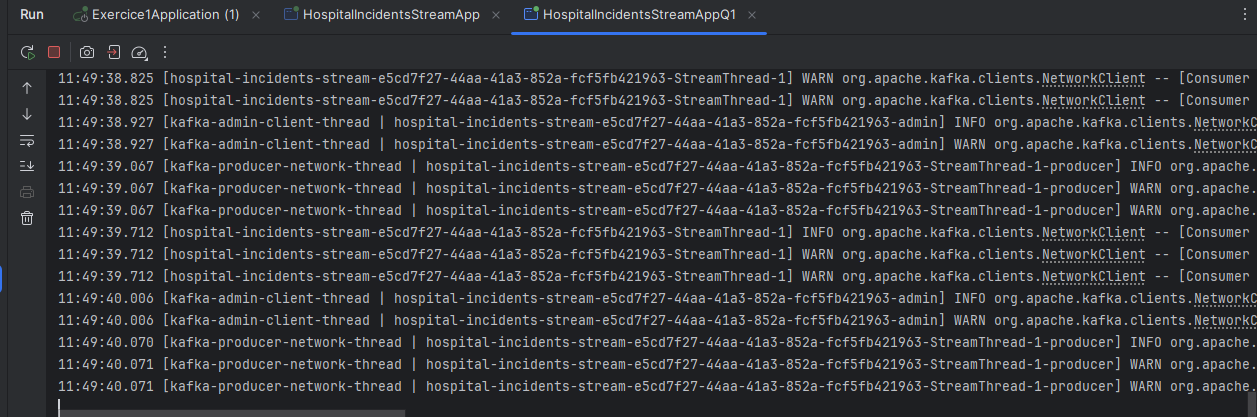


**Exercice 2:**

1. Développez une application Kafka Streams qui lit les données des incidents depuis un topic

Kafka nommé hospital\_incidents, filtre les incidents critiques (severity = "Critique"), classe les incidents par service et par type, puis agrège les résultats pour produire le nombre total d'incidents critiques par service et type, et envoie les résultats dans un topic Kafka nommé incident\_report.

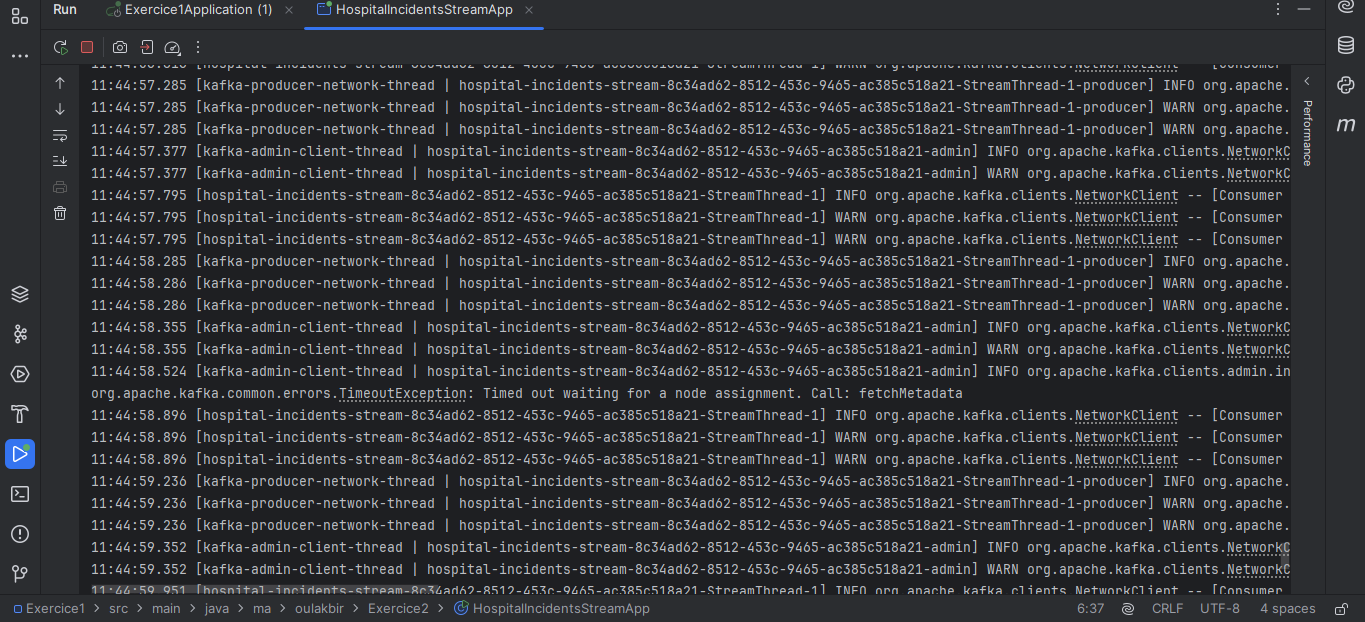
package ma.oulakbir.Exercice2;  
  
  
import org.apache.kafka.common.serialization.Serdes;  
import org.apache.kafka.streams.\*;  
import org.apache.kafka.streams.kstream.\*;  
import java.util.Properties;  
  
public class HospitalIncidentsStreamAppQ1 {  
  
 public static void main(String[] args) {  
 // Setup Kafka Streams properties  
 Properties props = new Properties();  
 props.put(StreamsConfig.*APPLICATION\_ID\_CONFIG*, "hospital-incidents-stream");  
 props.put(StreamsConfig.*BOOTSTRAP\_SERVERS\_CONFIG*, "localhost:9092");  
 props.put(StreamsConfig.*DEFAULT\_KEY\_SERDE\_CLASS\_CONFIG*, Serdes.*String*().getClass());  
 props.put(StreamsConfig.*DEFAULT\_VALUE\_SERDE\_CLASS\_CONFIG*, Serdes.*String*().getClass());  
  
 // Create a StreamsBuilder  
 StreamsBuilder builder = new StreamsBuilder();  
  
 // Step 1: Read from the 'hospital\_incidents' topic  
 KStream<String, String> incidentStream = builder.stream("hospital\_incidents");  
  
 // Step 2: Filter for critical incidents (severity = "Critique")  
 KStream<String, String> criticalIncidents = incidentStream.filter((key, value) -> {  
 String[] parts = value.split("\\|");  
 String severity = parts[2]; // Extract severity field  
 return severity.equals("Critique"); // Keep only critical incidents  
 });  
  
 // Step 3: Send the filtered incidents to a new topic 'critical\_incidents'  
 criticalIncidents.to("critical\_incidents"); // Write to Kafka topic  
  
 // Build and start the Kafka Streams application  
 KafkaStreams streams = new KafkaStreams(builder.build(), props);  
 Runtime.*getRuntime*().addShutdownHook(new Thread(streams::close));  
 streams.start();  
 }  
}



2. Modifiez l'application Kafka Streams pour calculer la durée moyenne entre deux incidents critiques pour chaque service. Configurez l'application pour produire un rapport dans un topic Kafka nommé incident\_time\_analysis, contenant la durée moyenne entre les incidents critiques par service.

package ma.oulakbir.Exercice2;  
  
import org.apache.kafka.common.serialization.Serdes;  
import org.apache.kafka.streams.\*;  
import org.apache.kafka.streams.kstream.\*;  
import org.apache.kafka.streams.state.Stores;  
import org.apache.kafka.streams.state.KeyValueStore;  
import org.apache.kafka.common.utils.Bytes;  
  
import java.text.SimpleDateFormat;  
import java.util.Date;  
import java.util.Properties;  
import java.util.concurrent.TimeUnit;  
  
public class HospitalIncidentsStreamApp {  
  
 public static void main(String[] args) {  
 // Setup Kafka Streams properties  
 Properties props = new Properties();  
 props.put(StreamsConfig.*APPLICATION\_ID\_CONFIG*, "hospital-incidents-stream");  
 props.put(StreamsConfig.*BOOTSTRAP\_SERVERS\_CONFIG*, "localhost:9092");  
 props.put(StreamsConfig.*DEFAULT\_KEY\_SERDE\_CLASS\_CONFIG*, Serdes.*String*().getClass());  
 props.put(StreamsConfig.*DEFAULT\_VALUE\_SERDE\_CLASS\_CONFIG*, Serdes.*String*().getClass());  
  
 // Create a StreamsBuilder  
 StreamsBuilder builder = new StreamsBuilder();  
  
 // Step 1: Read from the 'hospital\_incidents' topic  
 KStream<String, String> incidentStream = builder.stream("hospital\_incidents");  
  
 // Step 2: Filter for critical incidents (severity = "Critique")  
 KStream<String, String> criticalIncidents = incidentStream.filter((key, value) -> {  
 String[] parts = value.split("\\|");  
 String severity = parts[2]; // Extract severity field  
 return severity.equals("Critique");  
 });  
  
 // Step 3: Extract service and timestamp from the incident data  
 KStream<String, Long> serviceAndTimestamp = criticalIncidents.map((key, value) -> {  
 String[] parts = value.split("\\|");  
 String service = parts[1]; // Extract service field  
 String timestamp = parts[4]; // Extract timestamp field  
 try {  
 SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd'T'HH:mm:ss'Z'");  
 Date date = sdf.parse(timestamp);  
 long timestampInMillis = date.getTime(); // Convert to milliseconds  
 return KeyValue.*pair*(service, timestampInMillis); // Use service as key and timestamp as value  
 } catch (Exception e) {  
 e.printStackTrace();  
 return KeyValue.*pair*("UnknownService", 0L); // Return default in case of error  
 }  
 });  
  
 // Step 4: Group incidents by service  
 KGroupedStream<String, Long> groupedByService = serviceAndTimestamp.groupByKey();  
  
 // Step 5: Aggregate to calculate average time between incidents per service  
 KTable<String, Double> averageTimeBetweenIncidents = groupedByService.aggregate(  
 () -> 0.0, // Initial value: no time difference yet  
 (service, newTime, aggregate) -> {  
 long newTimeMillis = newTime;  
 double oldAverage = aggregate;  
 // If there is no previous incident, skip calculation  
 if (oldAverage == 0.0) {  
 return 0.0; // No previous incident, skip calculation  
 } else {  
 double timeDifference = (newTimeMillis - oldAverage) / 1000.0; // Time difference in seconds  
 return (newTimeMillis + timeDifference) / 2; // Return new average  
 }  
 },  
 Materialized.<String, Double, KeyValueStore<Bytes, byte[]>>*as*("incident-aggregates-store")  
 .withKeySerde(Serdes.*String*()) // Specify the key serializer (String)  
 .withValueSerde(Serdes.*Double*()) // Specify the value serializer (Double)  
 );  
  
 // Step 6: Format the time difference into hours and minutes  
 averageTimeBetweenIncidents.toStream().mapValues(value -> {  
 long averageTimeInSeconds = Math.*round*(value); // Round the Double to the nearest long  
 long hours = TimeUnit.*SECONDS*.toHours(averageTimeInSeconds);  
 long minutes = TimeUnit.*SECONDS*.toMinutes(averageTimeInSeconds) - TimeUnit.*HOURS*.toMinutes(hours);  
 return String.*format*("service: \"%s\", averageTimeBetweenIncidents : \"%dh %dm\"",  
 "service\_name\_placeholder", hours, minutes); // Placeholder for service name  
 }).to("incident\_time\_analysis"); // Write to Kafka topic  
  
 // Build and start the Kafka Streams application  
 KafkaStreams streams = new KafkaStreams(builder.build(), props);  
 Runtime.*getRuntime*().addShutdownHook(new Thread(streams::close));  
 streams.start();  
 }  
}

package ma.oulakbir.Exercice2;  
  
  
public class TimeTracker {  
 // Attributes for storing total duration, count, and last timestamp  
 public long totalDuration; // Total time between incidents  
 public long count; // Number of incidents  
 public long lastTimestamp; // Timestamp of the last incident  
 public String service; // The service for which we are tracking incidents  
  
 // Constructor to initialize the attributes  
 public TimeTracker(long totalDuration, long count, long lastTimestamp, String service) {  
 this.totalDuration = totalDuration;  
 this.count = count;  
 this.lastTimestamp = lastTimestamp;  
 this.service = service;  
 }  
  
 // Setter methods if needed (optional)  
 public void setTotalDuration(long totalDuration) {  
 this.totalDuration = totalDuration;  
 }  
  
 public void setCount(long count) {  
 this.count = count;  
 }  
  
 public void setLastTimestamp(long lastTimestamp) {  
 this.lastTimestamp = lastTimestamp;  
 }  
  
 public void setService(String service) {  
 this.service = service;  
 }  
  
 // Getter methods (optional)  
 public long getTotalDuration() {  
 return totalDuration;  
 }  
  
 public long getCount() {  
 return count;  
 }  
  
 public long getLastTimestamp() {  
 return lastTimestamp;  
 }  
  
 public String getService() {  
 return service;  
 }  
}



Application.properties file:

spring.application.name=Exercice1  
  
spring.datasource.url=jdbc:h2:mem:hospital-db  
spring.datasource.driver-class-name=org.h2.Driver  
spring.datasource.username=sa  
spring.datasource.password=  
spring.h2.console.enabled=true  
spring.h2.console.path=/h2-console  
server.port=8082