

**Pom packaging**

<packaging>pom</packaging>

Asta inseamna ca nu se va crea vreun jar sau war pentru project. Project defapt va contine alte proiecte sau modules in el, si acest pom.xml va fi doar comun pentru ele si gata.

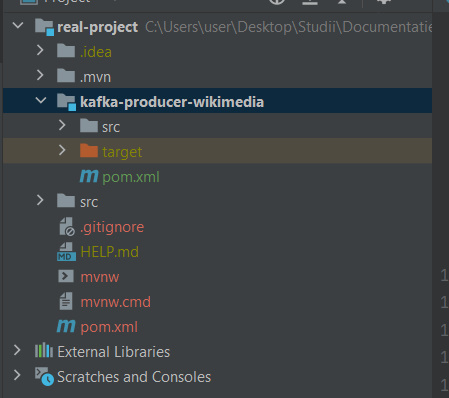
* Acum, cream in el un Module, ca Mavem quickstart archetype si vedem ca in pom.xml de baza a aparut asa

<modules>  
 <module>kafka-producer-wikimedia</module>  
</modules>

Module creat inca nu e un spring boot application, dar il facem usor. Ideea e ca pom.xml e simplu si oricum preia tot de la pom.xml parinte, unde am pus packaging ca pom.

- asta si e ideea. Facand un pom.xml global, toate microserviciile, sau mai bine zis modules, vor prelua toate dependentele si setarile de la el.

- asa nu mai e necesar sa le cream din start ca spring boot application, ci ca maven archetype





**Create Producer**

* Wikimedia are un REST API ce ofera date cu toate modificarile ce se fac in real time
* Modificarile se fac foarte des, si avem nevoie sa fac un producer care returneaza de fiecare data cand o modificare apare un nou mesaj catre Kafka
* Pentru asta, folosim OkHttp pentru a crea un sistem ce creaza un event odata ce wikimedia isi modifica datele.

1. **Adaugam dependentele pentru OkHttp si Jackson**

<dependency>  
 <groupId>com.squareup.okhttp3</groupId>  
 <artifactId>okhttp</artifactId>  
 <version>5.0.0-alpha.11</version>  
</dependency>  
  
<dependency>  
 <groupId>com.launchdarkly</groupId>  
 <artifactId>okhttp-eventsource</artifactId>  
 <version>2.5.0</version>  
</dependency>

Atentie la versiunea 2.5.0!!!

<dependency>  
 <groupId>com.fasterxml.jackson.core</groupId>  
 <artifactId>jackson-core</artifactId>  
</dependency>  
<dependency>  
 <groupId>com.fasterxml.jackson.core</groupId>  
 <artifactId>jackson-databind</artifactId>  
</dependency>

1. **Implementam EventHandler**

@RequiredArgsConstructor  
public class WikimediaChangesHandler implements EventHandler {  
 private static final Logger *LOGGER* = Logger.*getLogger*(WikimediaChangesHandler.class.getName());  
   
 private final String topic1;  
  
 private final KafkaTemplate<String, String> kafkaTemplate;  
  
 @Override  
 public void onOpen() throws Exception {  
  
 }  
  
 @Override  
 public void onClosed() throws Exception {  
  
 }  
  
 @Override  
 public void onMessage(String s, MessageEvent messageEvent) throws Exception {  
 *LOGGER*.info(String.*format*("event data -> $s",messageEvent.getData()));  
  
 kafkaTemplate.send(topic1,messageEvent.getData());  
 }  
  
 @Override  
 public void onComment(String s) throws Exception {  
  
 }  
  
 @Override  
 public void onError(Throwable throwable) {  
  
 }  
}

De fiecare data cand e un nou event pe wikimedia, metoda onMessage() va fi triggered si va citi acest nou event si deci vom inregistra aceste date cu Kafka in topic

1. **Si cream un Producer**

@Service  
@RequiredArgsConstructor  
public class WikimediaChangeProducer {  
 @Value("${spring.kafka.topic.name}")  
 private String topic1;  
  
 private static final Logger *LOGGER* = Logger.*getLogger*(WikimediaChangeProducer.class.getName());  
  
 private final KafkaTemplate<String, String > kafkaTemplate;  
  
 public void sendMessage() throws InterruptedException {  
 EventHandler eventHandler = new WikimediaChangesHandler(topic1,kafkaTemplate);  
 String url = "https://stream.wikimedia.org/v2/stream/recentchange";  
  
 EventSource eventSource = new EventSource.Builder(eventHandler, URI.*create*(url)).build();  
 eventSource.start();  
  
  
 }  
}

EventSource se conecteaza cu wikimedia si citeste toate datele

Daca ne uitam la EventSource class, ea are asa fields:

private final ExecutorService eventExecutor;  
private final ExecutorService streamExecutor;

Deci, ea foloseste ExecutorService, de aia si folosim start()

1. **Cand aplicatia va rula, va incepe sa trimita date de la wikimedia**

@SpringBootApplication  
public class SpringBootProducerApplication {  
 public static void main(String[] args) {  
 SpringApplication.*run*(SpringBootProducerApplication.class);  
 }  
 @Autowired  
 private WikimediaChangeProducer producer;  
 @Bean  
 public CommandLineRunner commandLineRunner(){  
 return new CommandLineRunner() {  
 @Override  
 public void run(String... args) throws Exception {  
 producer.sendMessage();  
 }  
 };  
 }  
}

Vedem ca in topic vin si vin date noi intr-una.

1. **Cream Consumer in alt microservice**

@Service  
public class KafkaDataBaseConsumer {  
 private final Logger logger = Logger.*getLogger*(KafkaDataBaseConsumer.class.getName());  
  
 @KafkaListener(groupId = "myGroup", topics = "wikimedia\_recentchange")  
 public void consume(String message){  
 logger.info("Message: "+message);  
 }  
}

1. **Cream entity pentru baza de date**

@Getter  
@Setter  
@Entity  
@Table(name = "wikimedia\_recentchange")  
public class WikimediaData {  
 @Id  
 @GeneratedValue(strategy = GenerationType.*IDENTITY*)  
 private Long id;  
  
 @Lob  
 private String wikiEventData;  
}

Deoarece datele vor fi foarte mari, vom avea o coloana de tip BLOB, si anotatia @Lob asta si spune

Cream repository si deja Consumer va fi asa:

@Service  
@RequiredArgsConstructor  
public class KafkaDataBaseConsumer {  
 private final WikimediaDataRepository dataRepository;  
 private final Logger logger = Logger.*getLogger*(KafkaDataBaseConsumer.class.getName());  
  
 @KafkaListener(groupId = "myGroup", topics = "wikimedia\_recentchange")  
 public void consume(String message){  
 logger.info("Message: "+message);  
  
 WikimediaData wikimediaData = new WikimediaData();  
 wikimediaData.setWikiEventData(message);  
  
 dataRepository.save(wikimediaData);  
 logger.info("Object successfully saved.");  
  
 }  
}