Merged Opened 2 months ago by Angelo Flores

Curated Collections - Lambda Processor

All threads resolved

Compare and

src/main/java/com/nordstrom/nse/curatedCollections/clients/AmazonS3Client.java +23 -0 Viewed

src/main/java/com/nordstrom/nse/curatedCollections/clients/S3Client.java +64 -0 Viewed

src/main/java/com/nordstrom/nse/curatedCollections/config/CompassConfiguration.java +22 -0 Viewed

```
1  + package com.nordstrom.nse.curatedCollections.config;
2  +
3  + public class CompassConfiguration {
4  +   private String s3Bucket;
5  +   private String uploadBucket;
6  +
7  +   public String getS3Bucket() {
8  +     return s3Bucket;
9  +   }
10 +
11 +   public void setS3Bucket(String s3Bucket) {
12 +     this.s3Bucket = s3Bucket;
13 +   }
14 +
15 +   public String getUploadBucket() {
16 +     return uploadBucket;
17 +   }
18 +
19 +   public void setUploadBucket(String uploadBucket) {
20 +     this.uploadBucket = uploadBucket;
21 +   }
22 + }
```

src/main/java/com/nordstrom/nse/curatedCollections/config/EnvironmentConfiguration.java +134 -0 Viewed

```
1  + package com.nordstrom.nse.curatedCollections.config;
2  +
3  + import com.amazonaws.services.s3.AmazonS3;
4  + import com.amazonaws.util.StringUtils;
5  + import com.nordstrom.nse.curatedCollections.Constants;
6  + import com.nordstrom.nse.curatedCollections.clients.AmazonS3Client;
7  + import com.nordstrom.nse.curatedCollections.clients.S3Client;
8  + import com.nordstrom.nse.curatedCollections.services.SSMService;
9  + import java.io.IOException;
10 + import java.util.Properties;
11 + import org.apache.avro.generic.GenericData.Record;
12 + import org.apache.kafka.clients.producer.KafkaProducer;
13 + import org.apache.log4j.Logger;
14 +
15 + public class EnvironmentConfiguration {
16 +   private static final Logger logger = Logger.getLogger(EnvironmentConfiguration.class);
17 +   private static EnvironmentConfiguration environmentConfiguration;
18 +   private final String COMMON_RESOURCE_FILE = "/curated-collections-%s.properties";
19 +   private final String RESOURCE_FILE_FORMAT = "/kafka-%s.properties";
20 +   private String environmentName;
21 +   private CompassConfiguration compassConfiguration;
22 +   private InferenceEngineConfiguration inferenceEngineConfiguration;
23 +   private S3Client s3Client;
24 +   private Properties props;
25 +   private String kafkaTopic;
26 +
27 +   public static EnvironmentConfiguration getInstance(String env) {
28 +     if (environmentConfiguration == null) {
29 +       synchronized (EnvironmentConfiguration.class) {
30 +         if (environmentConfiguration == null) {
31 +           String environmentName = env;
```

```
32  +
33  +           if (!StringUtils.isNullOrEmpty(env)) {
34  +             environmentName = System.getenv(Constants.APPLICATION_ENVIRONMENT);
35  +           }
36  +
37  +           environmentConfiguration = new EnvironmentConfiguration(environmentName);
38  +         }
39  +       }
40  +     }
41  +
42  +     return environmentConfiguration;
43  +   }
44  +
45  +   private EnvironmentConfiguration(String environmentName) {
46  +     setEnvironmentName(environmentName);
47  +     buildEnvironment();
48  +   }
49  +
50  +   private void buildEnvironment() {
51  +     try {
52  +       props = new Properties();
53  +       props.load(getClass().getResourceAsStream(String.format(COMMON_RESOURCE_FILE, environmentName)));
54  +
55  +       var compassConfiguration = new CompassConfiguration();
56  +       setCompassConfiguration(compassConfiguration);
57  +
58  +       var inferenceEngineConfiguration = new InferenceEngineConfiguration();
59  +       setInferenceEngineConfiguration(inferenceEngineConfiguration);
60  +
61  +       setS3Client(AmazonS3Client.getAmazonS3());
62  +     }
63  +     catch (Exception exception) {
64  +       logger.error("Failed to build environment: " + exception.toString());
65  +     }
66  +   }
67  +
68  +   public void setEnvironmentName(String environmentName) {
69  +     if(environmentName == null || StringUtils.isNullOrEmpty(environmentName)){
70  +       // For Testing
71  +       this.environmentName = "dev";
72  +     }
73  +     else {
74  +       this.environmentName = environmentName;
75  +     }
76  +   }
77  +
78  +   public KafkaProducer<String, Record> generateKafkaProducer() throws IOException {
79  +     props = new Properties();
80  +     props.load(getClass().getResourceAsStream(String.format(RESOURCE_FILE_FORMAT, environmentName)));
81  +
82  +     setJaasConfig();
83  +     setKafkaTopic();
84  +     return new KafkaProducer<>(props);
85  +   }
86  +
87  +   public void setCompassConfiguration(CompassConfiguration compassConfiguration) {
88  +     compassConfiguration.setUploadBucket(props.getProperty("tokenstyleupload.bucket"));
89  +     compassConfiguration.setS3Bucket(props.getProperty("curatedcollections.bucket"));
90  +     this.compassConfiguration = compassConfiguration;
91  +   }
92  +
93  +   private void setInferenceEngineConfiguration(InferenceEngineConfiguration inferenceEngineConfiguration)
    {
94  +     inferenceEngineConfiguration.setInferenceBucket(props.getProperty("inference.bucket"));
95  +     this.inferenceEngineConfiguration = inferenceEngineConfiguration;
96  +   }
97  +
98  +   private void setS3Client(AmazonS3 amazonS3) {
99  +     this.s3Client = new S3Client(amazonS3);
100 +   }
101 +
102 +   public void setS3Client(S3Client s3Client) {
103 +     // Testing
104 +     this.s3Client = s3Client;
105 +   }
```

```
+
+   private void setJaasConfig() {
+     var ssmService = new SSMService();
+     var config = props.getProperty("sasl.jaas.config");
+     var kafkaUsername = ssmService.getParameterStoreValue(props.getProperty("kafka.ssm.accesskey"));
+     var kafkaPassword = ssmService.getParameterStoreValue(props.getProperty("kafka.ssm.secretkey"));
+     props.setProperty("sasl.jaas.config", String.format(config, kafkaUsername, kafkaPassword));
+   }
+
+   private void setKafkaTopic() {
+     this.kafkaTopic = props.getProperty("kafka.topic");
+   }
+
+   public CompassConfiguration getCompassConfiguration() {
+     return compassConfiguration;
+   }
+
+   public InferenceEngineConfiguration getInferenceEngineConfiguration() {
+     return inferenceEngineConfiguration;
+   }
+
+   public S3Client getS3Client() {
+     return s3Client;
+   }
+
+   public String getKafkaTopic() {
+     return kafkaTopic;
+   }
+ }
```

src/main/java/com/nordstrom/nse/curatedCollections/config/InferenceEngineConfiguration.java +13 -0 Viewed

```
+ package com.nordstrom.nse.curatedCollections.config;
+
+ public class InferenceEngineConfiguration {
+   private String inferenceBucket;
+
+   public String getInferenceBucket() {
+     return inferenceBucket;
+   }
+
+   public void setInferenceBucket(String inferenceBucket) {
+     this.inferenceBucket = inferenceBucket;
+   }
+ }
  \ No newline at end of file
```

src/main/java/com/nordstrom/nse/curatedCollections/handlers/CuratedCollectionsHandler.java +115 -0 Viewed

```
+ package com.nordstrom.nse.curatedCollections.handlers;
+
+ import com.amazonaws.services.lambda.runtime.Context;
+ import com.amazonaws.services.lambda.runtime.RequestHandler;
+ import com.amazonaws.services.lambda.runtime.events.S3Event;
+ import com.amazonaws.services.s3.event.S3EventNotification.S3EventNotificationRecord;
+ import com.amazonaws.util.StringUtils;
+ import com.nordstrom.nse.curatedCollections.Constants;
+ import com.nordstrom.nse.curatedCollections.config.EnvironmentConfiguration;
+ import com.nordstrom.nse.curatedCollections.models.curatedcollection.CuratedCollectionsResponse;
+ import com.nordstrom.nse.curatedCollections.services.CuratedCollectionsService;
+ import java.util.Arrays;
+ import java.util.List;
+ import java.util.stream.Collectors;
+ import org.apache.log4j.Logger;
+ import static com.nordstrom.nse.curatedCollections.Constants.SUCCESS;
+ import static com.nordstrom.nse.curatedCollections.Constants.FAIL;
+ import static com.nordstrom.nse.curatedCollections.Constants.NULL_INPUT;
+ import static com.nordstrom.nse.curatedCollections.Constants.ZERO_RECORDS;
+
+ public class CuratedCollectionsHandler implements RequestHandler<S3Event, String> {
+     private static final Logger logger = Logger.getLogger(CuratedCollectionsHandler.class);
+     private static EnvironmentConfiguration environment;
+     private static CuratedCollectionsService curatedCollectionsService;
+
+     static {
```

```
27  +          environment =
    EnvironmentConfiguration.getInstance(System.getenv(Constants.APPLICATION_ENVIRONMENT));
28  +          curatedCollectionsService =
29  +              new CuratedCollectionsService(environment.getS3Client(),
30  +                  environment.getCompassConfiguration(), environment.getInferenceEngineConfiguration(),
    environment);
31  +      }
32  +
33  +      /**
34  +       * C'tor.
35  +       */
36  +      public String handleRequest(S3Event s3Event, Context context) {
37  +          logger.debug("CuratedCollectionsHandler Invoked with event: " + s3Event);
38  +          if (s3Event == null) {
39  +              logger.error(NULL_INPUT);
40  +              return NULL_INPUT;
41  +          }
42  +
43  +          List<S3EventNotificationRecord> records = s3Event.getRecords();
44  +
45  +          if (records == null) {
46  +              logger.error(NULL_INPUT);
47  +              return NULL_INPUT;
48  +          }
49  +
50  +          if (records.size() == 0) {
51  +              logger.error(ZERO_RECORDS);
52  +              return ZERO_RECORDS;
53  +          }
54  +
55  +          String bucket = s3Event.getRecords().get(0).getS3().getBucket().getName();
56  +          String key = s3Event.getRecords().get(0).getS3().getObject().getUrlDecodedKey();
57  +
58  +          if(uploadCuratedTokenStyleIds(key) != null){
59  +              return SUCCESS;
60  +          }
61  +          else{
62  +              return FAIL;
63  +          }
64  +      }
65  +
66  +      public CuratedCollectionsResponse uploadCuratedTokenStyleIds(String pathUrlFileName) {
67  +          CuratedCollectionsResponse curatedCollectionsResponse = null;
68  +          List<String> proposedStyleIds;
69  +
70  +          if (pathUrlFileName == null) {
71  +              logger.error("pathUrlFileName is null.");
72  +              return null;
73  +          }
74  +
75  +          String[] parts = pathUrlFileName.split("\\.");
76  +
77  +          String token = parts[0];
78  +          if (StringUtils.isNullOrEmpty(token)) {
79  +              logger.error("No tokens to update.");
80  +              return null;
81  +          }
82  +
83  +          try {
84  +              String tokenStyleContent = environment.getS3Client()
85  +                  .getObject(environment.getCompassConfiguration().getUploadBucket(), pathUrlFileName);
86  +              String[] splits = tokenStyleContent.split("\n");
87  +
88  +              proposedStyleIds = Arrays.asList(splits)
89  +                  .stream()
90  +                  .map(styleId -> styleId.trim())
91  +                  .collect(Collectors.toList());
92  +          } catch (Exception ex) {
93  +              logger.error("Empty Style IDs", ex);
94  +              return null;
95  +          }
96  +
97  +          try {
98  +              curatedCollectionsResponse = curatedCollectionsService.process(token, proposedStyleIds);
99  +
100 +              return curatedCollectionsResponse;
```

```
101 +         } catch (Exception ex) {
102 +             logger.error("Unable to create CuratedCollectionsResponse with exception", ex);
103 +             return null;
104 +         }
105 +     }
106 +
107 +     public void setCuratedCollectionsService(CuratedCollectionsService curatedCollections) {
108 +         curatedCollectionsService = curatedCollections;
109 +     }
110 +
111 +     public void setEnvironment(EnvironmentConfiguration env) {
112 +         environment = env;
113 +     }
114 + }
115 +
```

src/main/java/com/nordstrom/nse/curatedCollections/models/curatedcollection/CuratedCollectionsResponse.java +13 -0 Viewed

```
1  + package com.nordstrom.nse.curatedCollections.models.curatedcollection;
2  +
3  + public class CuratedCollectionsResponse {
4  +   private int filesUpdated;
5  +
6  +   public int getFilesUpdated() {
7  +     return filesUpdated;
8  +   }
9  +
10 +   public void setFilesUpdated(int filesUpdated) {
11 +     this.filesUpdated = filesUpdated;
12 +   }
13 + }
```

src/main/java/com/nordstrom/nse/curatedCollections/models/curatedcollection/InferenceSourceOutput.java +29 -0 Viewed

```
1  + package com.nordstrom.nse.curatedCollections.models.curatedcollection;
2  +
3  + import com.fasterxml.jackson.annotation.JsonProperty;
4  +
5  + import java.util.List;
6  +
7  + public class InferenceSourceOutput {
8  +   @JsonProperty("modified")
9  +   private String modified;
10 +
11 +   @JsonProperty("tokens")
12 +   private List<String> tokens;
13 +
14 +   public String getModified() {
15 +     return modified;
16 +   }
17 +
18 +   public void setModified(String modified) {
19 +     this.modified = modified;
20 +   }
21 +
22 +   public List<String> getTokens() {
23 +     return tokens;
24 +   }
25 +
26 +   public void setTokens(List<String> tokens) {
27 +     this.tokens = tokens;
28 +   }
29 + }
```

src/main/java/com/nordstrom/nse/curatedCollections/services/CuratedCollectionsService.java +211 -0 Viewed

```
1  + package com.nordstrom.nse.curatedCollections.services;
2  +
3  + import com.amazonaws.services.s3.model.AmazonS3Exception;
4  + import com.amazonaws.util.StringUtils;
5  + import com.fasterxml.jackson.core.JsonProcessingException;
6  + import com.fasterxml.jackson.databind.ObjectMapper;
7  + import com.nordstrom.nse.curatedCollections.clients.S3Client;
```

```
+ import com.nordstrom.nse.curatedCollections.config.CompassConfiguration;
+ import com.nordstrom.nse.curatedCollections.config.EnvironmentConfiguration;
+ import com.nordstrom.nse.curatedCollections.config.InferenceEngineConfiguration;
+ import com.nordstrom.nse.curatedCollections.models.curatedcollection.CuratedCollectionsResponse;
+ import com.nordstrom.nse.curatedCollections.models.curatedcollection.InferenceSourceOutput;
+ import com.nordstrom.nse.navigation.services.DateTimeService;
+ import java.io.IOException;
+ import java.util.ArrayList;
+ import java.util.Arrays;
+ import java.util.List;
+ import java.util.stream.Collectors;
+ import org.apache.avro.Schema;
+ import org.apache.avro.Schema.Parser;
+ import org.apache.avro.generic.GenericData.Record;
+ import org.apache.avro.generic.GenericRecordBuilder;
+ import org.apache.kafka.clients.producer.KafkaProducer;
+ import org.apache.kafka.clients.producer.ProducerRecord;
+ import org.slf4j.Logger;
+ import static org.slf4j.LoggerFactory.getLogger;
+
+ public class CuratedCollectionsService {
+   private static final String COMPASS_FORMAT = "source-algorithm-output/%s/common/compass.json";
+   private static final String CURRENT_TOKEN_STYLEID_FILENAME_FORMAT = "%s.csv";
+   private static final String PREVIOUS_TOKEN_STYLEID_FILENAME_FORMAT = "%s-previous.csv";
+   private final ObjectMapper objectMapper = new ObjectMapper();
+
+   private Logger logger = getLogger(CuratedCollectionsService.class);
+   private EnvironmentConfiguration environment;
+   private S3Client s3Client;
+   private CompassConfiguration compassConfiguration;
+   private InferenceEngineConfiguration inferenceEngineConfiguration;
+   private CuratedCollectionsResponse curatedCollectionsResponse;
+
+   public CuratedCollectionsService(S3Client s3Client, CompassConfiguration compassConfiguration,
+       InferenceEngineConfiguration inferenceEngineConfiguration, EnvironmentConfiguration environment) {
+     this.environment = environment;
+     this.s3Client = s3Client;
+     this.compassConfiguration = compassConfiguration;
+     this.inferenceEngineConfiguration = inferenceEngineConfiguration;
+     this.curatedCollectionsResponse = new CuratedCollectionsResponse();
+   }
+
+   public CuratedCollectionsResponse process(String token, List<String> proposedStyleIds) throws
IOException {
+     try {
+       logger.info("process: token={} proposedStyleIds={}", token, String.join(",", proposedStyleIds));
+
+       String currentTokenStyleIdsFileName = String.format(CURRENT_TOKEN_STYLEID_FILENAME_FORMAT, token);
+
+       // Get previous s3 token-styleids (navigation-curated-collections-prod/{token}.csv)
+       String currentContent = null;
+       List<String> currentStyleIds = Arrays.asList();
+
+       try {
+         currentContent = s3Client.getObject(compassConfiguration.getS3Bucket(),
currentTokenStyleIdsFileName);
+         logger.info("process: token={} currentContent={}", token, currentContent);
+
+         currentStyleIds = splitStyleIds(currentContent);
+       } catch (AmazonS3Exception ex) {
+         logger.info(String.format("process: No content found for file=%s, error=%s",
currentTokenStyleIdsFileName, ex.getMessage()));
+       }
+
+       // Generate a diff (added and removed styleIds for the token)
+       List<String> finalCurrentStyleIds = currentStyleIds;
+       List<String> addedStyleIds = proposedStyleIds.stream()
+           .distinct()
+           .filter(styleId -> !finalCurrentStyleIds.contains(styleId))
+           .collect(Collectors.toList());
+
+       List<String> removedStyleIds = currentStyleIds.stream()
+           .distinct()
+           .filter(styleId -> !proposedStyleIds.contains(styleId))
+           .collect(Collectors.toList());
```

```
+
+       logger.info("process: token={} addedStyles={}", token, String.join(",", addedStyleIds));
+       logger.info("process: token={} removedStyles={}", token, String.join(",", removedStyleIds));
+
+       // Update s3 source-algorithm-output/{styleIds}/common/compass.json for all related styleIds (adding/removing token)
+       addedStyleIds.forEach(styleId -> {
+         try {
+           addToken(styleId.trim(), token);
+         } catch (IOException exception) {
+           logger.error("Error adding token: " + exception);
+         }
+       });
+
+       removedStyleIds.forEach(styleId -> {
+         try {
+           removeToken(styleId.trim(), token);
+         } catch (IOException exception) {
+           logger.error("Error removing token: " + exception);
+         }
+       });
+
+       // Move old token-stylegroups: navigation-curated-collections-prod/{token}.csv -> navigation-curated-collections-prod/{token}-previous.csv
+       if (!StringUtils.isNullOrEmpty(currentContent)) {
+         String previousTokenStyleIdsFileName = String.format(PREVIOUS_TOKEN_STYLEID_FILENAME_FORMAT, token);
+         s3Client.putObject(compassConfiguration.getS3Bucket(), previousTokenStyleIdsFileName, currentContent);
+       }
+
+       // Save proposed token-stylegroups  (navigation-curated-collections-prod/{token}.csv)
+       saveNewTokens(token, proposedStyleIds);
+
+       streamEvents(addedStyleIds, removedStyleIds);
+       curatedCollectionsResponse.setFilesUpdated(addedStyleIds.size() + removedStyleIds.size());
+
+       return curatedCollectionsResponse;
+     } catch (Exception ex) {
+       logger.error(String.format("Error processing token=[%s]", token), ex);
+       throw ex;
+     }
+   }
+
+   private List<String> splitStyleIds(String content) {
+     String[] splits = content.split("\n");
+     return Arrays.asList(splits);
+   }
+
+   private void addToken(String styleId, String token) throws IOException {
+     String inferenceSourceOutputFileName = String.format(COMPASS_FORMAT, styleId);
+
+     InferenceSourceOutput compassData = getInferenceData(inferenceSourceOutputFileName);
+
+     // add token
+     if (!compassData.getTokens().contains(token)) {
+       compassData.getTokens().add(token);
+       logger.info("addToken: styleId={} token={}", styleId, token);
+
+       putInferenceData(inferenceSourceOutputFileName, compassData);
+     }
+   }
+
+   private void removeToken(String styleId, String token) throws IOException {
+     String inferenceSourceOutputFileName = String.format(COMPASS_FORMAT, styleId);
+
+     InferenceSourceOutput compassData = getInferenceData(inferenceSourceOutputFileName);
+
+     // remove token
+     if (compassData.getTokens().contains(token)) {
+       compassData.getTokens().remove(token);
+
+       logger.info("removeToken: styleId={} token={}", styleId, token);
+       putInferenceData(inferenceSourceOutputFileName, compassData);
+     }
+   }
```

```
152 +
153 +   private void saveNewTokens(String token, List<String> proposedStyleIds) {
154 +     String currentTokenStyleIdsFileName = String.format(CURRENT_TOKEN_STYLEID_FILENAME_FORMAT, token);
155 +
156 +     String content = String.join(System.lineSeparator(), proposedStyleIds);
157 +     s3Client.putObject(compassConfiguration.getS3Bucket(), currentTokenStyleIdsFileName, content);
158 +   }
159 +
160 +   private void putInferenceData(String compassFilename, InferenceSourceOutput compassData) throws
    JsonProcessingException {
161 +     DateTimeService dateTimeService = new DateTimeService();
162 +     compassData.setModified(dateTimeService.getDateTimeNowUtcAsString());
163 +
164 +     String content = objectMapper.writeValueAsString(compassData);
165 +     s3Client.putObject(inferenceEngineConfiguration.getInferenceBucket(), compassFilename, content);
166 +   }
167 +
168 +   private InferenceSourceOutput getInferenceData(String compassFilename) throws IOException {
169 +     InferenceSourceOutput compassData;
170 +     try {
171 +       String content = s3Client.getObject(inferenceEngineConfiguration.getInferenceBucket(),
    compassFilename);
172 +
173 +       // parse file
174 +       compassData = objectMapper.readValue(content, InferenceSourceOutput.class);
175 +     } catch (AmazonS3Exception ex) {
176 +       logger.info(String.format("File not found: %s", compassFilename));
177 +       compassData = new InferenceSourceOutput();
178 +       compassData.setTokens(new ArrayList<>());
179 +     }
180 +     catch (IOException ex) {
181 +       logger.error(String.format("Error parsing %s", compassFilename), ex);
182 +       throw ex;
183 +     }
184 +
185 +     return compassData;
186 +   }
187 +
188 +   private void producerEvent(KafkaProducer<String, Record> producer, String partition, Record styleId) {
189 +     logger.info("Topic: " + environment.getKafkaTopic() + " StyleId: " + styleId);
190 +     producer
191 +         .send(new ProducerRecord<>(environment.getKafkaTopic(), partition, styleId));
192 +   }
193 +
194 +   public void streamEvents(List<String> addedStyleIds, List<String> removedStyleIds) throws IOException {
195 +     var producer = environment.generateKafkaProducer();
196 +     var stream = getClass().getResourceAsStream("/CuratedCollectionEvent.avsc");
197 +     Schema schema = new Parser().parse(stream);
198 +
199 +     for(String added: addedStyleIds) {
200 +       Record value = new GenericRecordBuilder(schema).set("style_id", added).build();
201 +       producerEvent(producer, added, value);
202 +     }
203 +     for(String removed: removedStyleIds) {
204 +       Record value = new GenericRecordBuilder(schema).set("style_id", removed).build();
205 +       producerEvent(producer, removed, value);
206 +     }
207 +
208 +     producer.flush();
209 +     producer.close();
210 +   }
211 + }
```

src/main/java/com/nordstrom/nse/curatedCollections/services/SSMService.java +44 -0 Viewed

src/main/java/com/nordstrom/nse/curatedCollections/Application.java +46 -47 Viewed

src/main/java/com/nordstrom/nse/curatedCollections/Constants.java +1 -0 Viewed

↕ Show all unchanged lines ↥ Show 20 lines

```
6  6   public static final String NULL_RECORDS = "NULL RECORDS";
7  7   public static final String ZERO_RECORDS = "ZERO RECORDS";
8  8   public static final String SUCCESS = "SUCCESS";
   9 + public static final String FAIL = "RESPONSE FAILURE";
9  10 }
```

src/main/java/com/nordstrom/nse/curatedCollections/CuratedCollectionsHandler.java deleted +0 -19 Viewed

src/main/resources/CuratedCollectionEvent.avsc +11 -0 Viewed

```
   1 + {
   2 +   "type": "record",
   3 +   "name": "CuratedCollectionEvent",
   4 +   "namespace": "com.nordstrom.navigation",
   5 +   "fields": [
   6 +     {
   7 +       "name": "style_id",
   8 +       "type": "string"
   9 +     }
  10 +   ]
  11 + }
     \ No newline at end of file
```

src/main/resources/curated-collections-dev.properties +3 -0 Viewed

src/main/resources/curated-collections-prod.properties +3 -0 Viewed

src/main/resources/curated-collections.properties +0 -0 Viewed

src/main/resources/kafka-dev.properties +14 -0 Viewed

src/main/resources/kafka-prod.properties +14 -0 Viewed

src/test/java/com/nordstrom/nse/CuratedCollections/handlers/CuratedCollectionsHandlerTest.java +92 -0 Viewed

```
   1 + package com.nordstrom.nse.CuratedCollections.handlers;
   2 +
   3 + import com.nordstrom.nse.curatedCollections.clients.S3Client;
   4 + import com.nordstrom.nse.curatedCollections.config.CompassConfiguration;
   5 + import com.nordstrom.nse.curatedCollections.config.EnvironmentConfiguration;
   6 + import com.nordstrom.nse.curatedCollections.handlers.CuratedCollectionsHandler;
   7 + import com.nordstrom.nse.curatedCollections.services.CuratedCollectionsService;
   8 + import java.io.IOException;
   9 + import java.util.Properties;
  10 + import org.junit.Assert;
  11 + import static org.junit.Assert.assertEquals;
  12 + import org.junit.Before;
  13 + import org.junit.Test;
  14 + import org.mockito.Mock;
  15 + import static org.mockito.Mockito.when;
  16 + import org.mockito.MockitoAnnotations;
  17 + import org.mockito.stubbing.Answer;
  18 +
  19 + public class CuratedCollectionsHandlerTest {
  20 +   private static EnvironmentConfiguration environment;
  21 +   private final String COMMON_RESOURCE_FILE = "/curated-collections-dev.properties";
  22 +   private final String TEST_KEY = "back-to-school.csv";
  23 +   private Properties props;
  24 +
  25 +   static {
  26 +     environment = EnvironmentConfiguration.getInstance("dev");
  27 +   }
  28 +
  29 +   @Mock
```

```
30  +   private S3Client s3Client;
31  +   @Mock
32  +   private CuratedCollectionsService curatedCollectionsService;
33  +   @Mock
34  +   private CompassConfiguration compassConfiguration;
35  +   @Mock
36  +   private CuratedCollectionsHandler curatedCollectionsHandler;
37  +
38  +   @Before
39  +   public void before(){
40  +     MockitoAnnotations.openMocks(this);
41  +     compassConfiguration = new CompassConfiguration();
42  +     curatedCollectionsHandler = new CuratedCollectionsHandler();
43  +   }
44  +
45  +   @Test(expected = AssertionError.class)
46  +   public void testUploadCuratedTokenStyleIds() throws IOException {
47  +     props = new Properties();
48  +     props.load(getClass().getResourceAsStream(COMMON_RESOURCE_FILE));
49  +
50  +     String tokenUploadBucket = props.getProperty("tokenstyleupload.bucket");
51  +     compassConfiguration.setUploadBucket(tokenUploadBucket);
52  +     environment.setS3Client(s3Client);
53  +     environment.setCompassConfiguration(compassConfiguration);
54  +
55  +     curatedCollectionsHandler.setCuratedCollectionsService(curatedCollectionsService);
56  +     curatedCollectionsHandler.setEnvironment(environment);
57  +
58  +     when(s3Client.getObject(tokenUploadBucket, TEST_KEY))
59  +         .thenReturn(getStyleIds())
60  +         .then((Answer<Boolean>) invocation -> {
61  +           Object[] args = invocation.getArguments();
62  +           var tokenUpload = args[0];
63  +           var key = args[1];
64  +
65  +           assertEquals(String.format("navigation-token-style-upload-%s", "dev"), tokenUpload);
66  +           assertEquals(TEST_KEY, key);
67  +
68  +           return true;
69  +         });
70  +
71  +     curatedCollectionsHandler.uploadCuratedTokenStyleIds(TEST_KEY);
72  +     Assert.fail();
73  +   }
74  +
75  +   private String getStyleIds() {
76  +     return "5737150\n"
77  +         + "5309092\n"
78  +         + "5677036\n"
79  +         + "5951543\n"
80  +         + "5224682\n"
81  +         + "5915905\n"
82  +         + "5522645\n"
83  +         + "5796491\n"
84  +         + "5913041\n"
85  +         + "5542383\n"
86  +         + "5874746\n"
87  +         + "5737158\n"
88  +         + "4655774\n"
89  +         + "5935793\n"
90  +         + "4358682\n";
91  +   }
92  + }
```

src/test/java/com/nordstrom/nse/CuratedCollections/services/CuratedCollectionsServiceTest.java +108 -0 Viewed

```
1  + package com.nordstrom.nse.CuratedCollections.services;
2  +
3  + import com.nordstrom.nse.curatedCollections.clients.S3Client;
4  + import com.nordstrom.nse.curatedCollections.config.CompassConfiguration;
5  + import com.nordstrom.nse.curatedCollections.config.EnvironmentConfiguration;
6  + import com.nordstrom.nse.curatedCollections.config.InferenceEngineConfiguration;
7  + import com.nordstrom.nse.curatedCollections.models.curatedcollection.CuratedCollectionsResponse;
8  + import com.nordstrom.nse.curatedCollections.services.CuratedCollectionsService;
```

```
+ import java.io.IOException;
+ import java.util.Arrays;
+ import java.util.List;
+ import org.junit.Assert;
+ import static org.junit.Assert.assertEquals;
+ import org.junit.Before;
+ import org.junit.Test;
+ import org.mockito.Mock;
+ import static org.mockito.Mockito.when;
+ import org.mockito.MockitoAnnotations;
+
+ public class CuratedCollectionsServiceTest {
+   private static EnvironmentConfiguration environment;
+   private final String S3_BUCKET = "curatedcollections.bucket.dev";
+   private final String INFERENCE_BUCKET = "inference-engine-dev";
+   private final String UPLOAD_BUCKET = "navigation-token-style-upload-dev";
+
+   static {
+     environment = EnvironmentConfiguration.getInstance("prod");
+   }
+
+   @Mock
+   private CuratedCollectionsService curatedCollectionsService;
+   @Mock
+   private CuratedCollectionsResponse curatedCollectionsResponse;
+   @Mock
+   private S3Client s3Client;
+   @Mock
+   private CompassConfiguration compassConfiguration;
+   @Mock
+   private InferenceEngineConfiguration inferenceEngineConfiguration;
+
+   @Before
+   public void before(){
+     MockitoAnnotations.openMocks(this);
+     compassConfiguration = new CompassConfiguration();
+     inferenceEngineConfiguration = new InferenceEngineConfiguration();
+     compassConfiguration.setS3Bucket(S3_BUCKET);
+     compassConfiguration.setUploadBucket(UPLOAD_BUCKET);
+     inferenceEngineConfiguration.setInferenceBucket(INFERENCE_BUCKET);
+     curatedCollectionsService = new
CuratedCollectionsService(s3Client,compassConfiguration,inferenceEngineConfiguration, environment);
+   }
+
+   @Test(expected = Exception.class)
+   public void testProcess() throws IOException {
+     String token = "back-to-school";
+     String compassData = "source-algorithm-output/1111111/common/compass.json";
+
+     when(s3Client.getObject(S3_BUCKET, token + ".csv"))
+         .thenReturn(getStyleIds() + "1111111");
+     when(s3Client.getObject(INFERENCE_BUCKET, compassData))
+         .thenReturn(getInferenceObject());
+     when(s3Client.putObject(INFERENCE_BUCKET, compassData, getInferenceObject()))
+         .thenReturn(true);
+     when(s3Client.putObject(S3_BUCKET, token + ".csv", getInferenceObject()))
+         .thenReturn(true);
+
+     curatedCollectionsResponse = curatedCollectionsService.process(token, getStyleIdList());
+
+     assertEquals(curatedCollectionsResponse.getFilesUpdated(), 1);
+     Assert.fail();
+   }
+
+   private String getInferenceObject() {
+     return "{\n"
+         + "  \"modified\": \"2021-02-24T16:05:34Z\",\n"
+         + "  \"tokens\": [\n"
+         + "    \"brands\",\n"
+         + "    \"cole-haan--173\",\n"
+         + "    \"browse\",\n"
+         + "    \"men\",\n"
+         + "    \"all\",\n"
+         + "    \"shoes\",\n"
+         + "    \"loafers-slip-ons\"\n"
+         + "  ]\n"
```

```
84  +          + "}";
85  +    }
86  +
87  +    private String getStyleIds() {
88  +      return "5737150\n"
89  +          + "5309092\n"
90  +          + "5677036\n"
91  +          + "5951543\n"
92  +          + "5224682\n"
93  +          + "5915905\n"
94  +          + "5522645\n"
95  +          + "5796491\n"
96  +          + "5913041\n"
97  +          + "5542383\n"
98  +          + "5874746\n"
99  +          + "5737158\n"
100 +          + "4655774\n"
101 +          + "5935793\n"
102 +          + "4358682\n";
103 +    }
104 +
105 +    private List<String> getStyleIdList() {
106 +      return Arrays.asList(getStyleIds().split("\n"));
107 +    }
108 + }
```

build.gradle +6 -1 Viewed