**Component Unit Testing**

Contents

[1. Component 1 2](#_Toc521502941)

[2. Component 2 4](#_Toc521502942)

[3. Component 3 6](#_Toc521502943)

1. **Component 1**

**RabbitMQ Sender component :**

|  |
| --- |
| import org.slf4j.Logger;  import org.slf4j.LoggerFactory;  import org.springframework.amqp.rabbit.connection.ConnectionFactory;  import org.springframework.amqp.rabbit.core.RabbitTemplate;  import org.springframework.beans.factory.annotation.Autowired;  import org.springframework.stereotype.Component;  import com.nbcu.tvrocs.common.constant.MQConstants;  import com.nbcu.tvrocs.model.status.LMWStatus;  @Component  public class RabbitMQSender {  static final Logger logger = LoggerFactory.getLogger(RabbitMQSender.class);  @Autowired  RabbitTemplate rabbitTemplate;  @Autowired  ConnectionFactory connectionFactory;  /\*\*  \* Object in Status Service Queue Creation Date : Mar 28, 2018 , 3:04:20 PM  \*/  public void statusServiceUpdate(LMWStatus lmwStatus) {  rabbitTemplate.convertAndSend(MQConstants.EXCHANGE\_STATUS\_SERVICE, MQConstants.KEY\_STATUS\_SERVICE, lmwStatus);  logger.info(  "Method Name :: statusServiceUpdate Request ID :: NA Execution ID :: {} Message :: Message sent to the status service queue successfully",  lmwStatus.getStageStatus().getExecutionId());  }  } |

**RabbitMQ Sender test:**

|  |
| --- |
| import static org.mockito.Mockito.doNothing;  import org.junit.Before;  import org.junit.Test;  import org.mockito.InjectMocks;  import org.mockito.Mock;  import org.mockito.Mockito;  import org.mockito.MockitoAnnotations;  import org.springframework.amqp.rabbit.core.RabbitTemplate;  import org.springframework.boot.test.context.SpringBootTest;  import com.nbcu.tvrocs.model.status.LMWStatus;  import com.nbcu.tvrocs.model.status.StageStatus;  import com.nbcu.tvrocs.reportengine.LMWReportEngineApplication;  @SpringBootTest(classes = LMWReportEngineApplication.class)  public class RabbitMqSenderTest {  @Mock  RabbitTemplate rabbitmq;  @InjectMocks  RabbitMQSender rabbitMQSender;  @Before  public void setUp() throws Exception {  MockitoAnnotations.initMocks(this);  }  @Test  public void statusServiceUpdateTest() {  LMWStatus lmwStatus = new LMWStatus();  StageStatus stageStatus = new StageStatus();  stageStatus.setExecutionId("1234");  lmwStatus.setStageStatus(stageStatus);  doNothing().when(rabbitmq).convertAndSend(Mockito.any(String.class), Mockito.any(String.class),  Mockito.any(LMWStatus.class));  rabbitMQSender.statusServiceUpdate(lmwStatus);  }  } |

1. **Component 2**

**RabbitMQ Listener component**

|  |
| --- |
| **import org.apache.commons.lang3.StringUtils;**  **import org.slf4j.Logger;**  **import org.slf4j.LoggerFactory;**  **import org.springframework.beans.factory.annotation.Autowired;**  **import org.springframework.stereotype.Service;**  **import com.nbcu.tvrocs.reportengine.exception.ReportEngineException;**  **import com.nbcu.tvrocs.reportengine.service.ReportEngineService;**  **import com.nbcu.tvrocs.reportengine.service.StatusService;**  **import com.nbcu.tvrocs.reportengine.utils.ReportEngineConstants;**  **import com.nbcu.tvrocs.reportengine.utils.ReportEngineUtil;**  **@Service**  **public class RabbitMqListner {**  **static final Logger logger = LoggerFactory.getLogger(RabbitMqListner.class);**  **@Autowired**  **RabbitMQSender rabbitMqSender;**  **@Autowired**  **StatusService statusService;**  **@Autowired**  **ReportEngineService reportEngineService;**  **/\*\***  **\* Method Name : reportEngineRabbitMQListner Description : Listen the Execution**  **\* Id from the Queue Creation Date : Mar 28, 2018 , 1:07:46 PM Created By :**  **\* bv115775**  **\*/**  **public void reportEngineRabbitMQListner(String executionId) throws ReportEngineException {**  **logger.info(**  **"Method Name :: reportEngineRabbitMQListner Execution ID :: {} Message :: Received the ExecutionId",**  **executionId);**  **try {**  **if (StringUtils.isBlank(executionId)) {**  **throw new ReportEngineException(ReportEngineConstants.BLANK\_REQUEST);**  **} else {**  **logger.info(**  **"Method Name :: reportEngineRabbitMQListner Execution ID :: {} Message :: Received the ExecutionId",**  **executionId);**  **rabbitMqSender.statusServiceUpdate(statusService.reportEngineStartStatus(executionId));**  **reportEngineService.reportGenerationService(executionId);**  **}**  **} catch (Exception de) {**  **logger.error(**  **"Method Name :: reportEngineRabbitMQListner Request ID :: NA Execution ID :: {} Error Message :: {} ",**  **executionId, ReportEngineUtil.convertStackTraceToString(de));**  **String failMessage = de.getMessage();**  **failMessage = StringUtils.isNotBlank(failMessage) && failMessage.length() > 2000**  **? failMessage.substring(0, 1999)**  **: failMessage;**  **rabbitMqSender.statusServiceUpdate(statusService.reportEngineErrorStatus(executionId, failMessage));**  **}**  **}**  **}** |

1. **Component 3**

|  |
| --- |
| **import static com.nbcu.tvrocs.reportengine.utils.ReportElementsConstant.RPT\_PROPERTY\_DEMOGRP;**  **import java.util.ArrayList;**  **import java.util.Collections;**  **import java.util.Comparator;**  **import java.util.LinkedList;**  **import java.util.List;**  **import java.util.Map;**  **import java.util.Map.Entry;**  **import java.util.Set;**  **import org.springframework.stereotype.Component;**  **import org.springframework.util.CollectionUtils;**  **import com.nbcu.tvrocs.reportengine.model.DataElementsWithColumns;**  **import com.nbcu.tvrocs.reportengine.utils.ReportEngineConstants;**  **@Component**  **public class CartesianProduct {**  **/\*\***  **\* Method Name : cartesianProduct Description : It will create the Cartesian**  **\* product from List of data Creation Date : Mar 9, 2018 , 3:41:28 PM Created By**  **\* : BV115775**  **\*/**  **protected static <T> List<List<T>> cartesianProduct(List<List<T>> lists) {**  **List<List<T>> resultLists = new ArrayList<>();**  **if (CollectionUtils.isEmpty(lists)) {**  **resultLists.add(new ArrayList<T>());**  **return resultLists;**  **} else {**  **List<T> firstList = lists.get(0);**  **List<List<T>> remainingLists = cartesianProduct(lists.subList(1, lists.size()));**  **for (T condition : firstList) {**  **for (List<T> remainingList : remainingLists) {**  **ArrayList<T> resultList = new ArrayList<>();**  **resultList.add(condition);**  **resultList.addAll(remainingList);**  **resultLists.add(resultList);**  **}**  **}**  **}**  **return resultLists;**  **}**  **/\*\***  **\* Method Name : createAcrossMetricCartesianProduct Description : This Method**  **\* will create cartesianProduct of across and metric Element Creation Date : Mar**  **\* 9, 2018 , 3:45:07 PM Created By : BV115775**  **\*/**  **public List<String> aCrossMetricCartesianProduct(Map<String, Map<String, List<String>>> unqiueDataElementList) {**  **List<List<String>> acrossMetricCartesianProductList;**  **List<String> acrossMetrics = new ArrayList<>();**  **if (!CollectionUtils.isEmpty(unqiueDataElementList.get(ReportEngineConstants.ACROSS))**  **&& !CollectionUtils.isEmpty(unqiueDataElementList.get(ReportEngineConstants.METRICS))) {**  **acrossMetricCartesianProductList = aCrossMetricsCartesianWithCombinedData(unqiueDataElementList);**  **} else {**  **acrossMetricCartesianProductList = aCrossMetricsCartesianByUniqueData(unqiueDataElementList);**  **Collections.sort(acrossMetricCartesianProductList, Comparator.comparing(e -> e.get(0)));**  **}**  **StringBuilder tempString = new StringBuilder();**  **tempString.append(ReportEngineConstants.EMPTY);**  **for (List<String> productElement : acrossMetricCartesianProductList) {**  **for (String s : productElement) {**  **tempString.append(s);**  **tempString.append(ReportEngineConstants.STRING\_SEPERATOR\_RE);**  **}**  **acrossMetrics.add(tempString.toString().substring(0, tempString.toString().length() - 1));**  **tempString = new StringBuilder();**  **tempString.append(ReportEngineConstants.EMPTY);**  **}**  **return acrossMetrics;**  **}**  **public List<List<String>> aCrossMetricsCartesianByUniqueData(**  **Map<String, Map<String, List<String>>> unqiueDataElementList) {**  **ArrayList<List<String>> acrossMetricDataList = new ArrayList<>();**  **List<String> metricsList = new ArrayList<>();**  **if (!CollectionUtils.isEmpty(unqiueDataElementList.get(ReportEngineConstants.ACROSS))) {**  **for (Entry<String, List<String>> acrossElementsKey : unqiueDataElementList.get(ReportEngineConstants.ACROSS)**  **.entrySet()) {**  **acrossMetricDataList**  **.add(unqiueDataElementList.get(ReportEngineConstants.ACROSS).get(acrossElementsKey.getKey()));**  **}**  **}**  **if (!CollectionUtils.isEmpty(unqiueDataElementList.get(ReportEngineConstants.METRICS))) {**  **for (Entry<String, List<String>> metricElementsKey : unqiueDataElementList**  **.get(ReportEngineConstants.METRICS).entrySet()) {**  **metricsList.add(metricElementsKey.getKey());**  **}**  **acrossMetricDataList.add(metricsList);**  **}**  **return cartesianProduct(acrossMetricDataList);**  **}**  **public List<List<String>> aCrossMetricsCartesianWithCombinedData(**  **Map<String, Map<String, List<String>>> unqiueDataElementList) {**  **List<List<String>> acrossMetricCartesianProductList = new ArrayList<>();**  **for (Entry<String, List<String>> metricElementsKey : unqiueDataElementList.get(ReportEngineConstants.METRICS)**  **.entrySet()) {**  **List<String> metricsList = new ArrayList<>();**  **metricsList.add(metricElementsKey.getKey());**  **ArrayList<List<String>> acrossMetricDataList = new ArrayList<>();**  **for (Entry<String, List<String>> acrossElementsKey : unqiueDataElementList.get(ReportEngineConstants.ACROSS)**  **.entrySet()) {**  **if (RPT\_PROPERTY\_DEMOGRP.equals(acrossElementsKey.getKey())) {**  **acrossMetricDataList.add(metricElementsKey.getValue());**  **} else {**  **acrossMetricDataList.add(**  **unqiueDataElementList.get(ReportEngineConstants.ACROSS).get(acrossElementsKey.getKey()));**  **}**  **}**  **acrossMetricDataList.add(metricsList);**  **acrossMetricCartesianProductList.addAll(cartesianProduct(acrossMetricDataList));**  **}**  **Collections.sort(acrossMetricCartesianProductList, Comparator.comparing(e -> e.get(0)));**  **return acrossMetricCartesianProductList;**  **}**  **public List<DataElementsWithColumns> combiningDataElementForReportLayout(**  **List<List<DataElementsWithColumns>> allReportParameter) {**  **List<DataElementsWithColumns> allReportParameterData = new LinkedList<>();**  **for (List<DataElementsWithColumns> reportParameterData : allReportParameter) {**  **for (DataElementsWithColumns dataElementsWithColumns : reportParameterData) {**  **allReportParameterData.add(dataElementsWithColumns);**  **}**  **}**  **return allReportParameterData;**  **}**  **/\*\***  **\* Method Name : createCartesianProduct Description : This method will create**  **\* Cartesian product of Down or across or section or metrics Creation Date : Mar**  **\* 9, 2018 , 3:46:01 PM Created By : BV115775**  **\*/**  **public List<String> createCartesianProduct(Map<String, List<String>> uniqueElementsData) {**  **List<List<String>> cartesianProductList;**  **List<String> downSection = new ArrayList<>();**  **ArrayList<List<String>> dataList = new ArrayList<>();**  **if (!CollectionUtils.isEmpty(uniqueElementsData)) {**  **for (Entry<String, List<String>> metricElementsKey : uniqueElementsData.entrySet()) {**  **dataList.add(uniqueElementsData.get(metricElementsKey.getKey()));**  **}**  **}**  **cartesianProductList = cartesianProduct(dataList);**  **if (org.apache.commons.collections.CollectionUtils.isNotEmpty(dataList)) {**  **StringBuilder tempString = new StringBuilder();**  **tempString.append(ReportEngineConstants.EMPTY);**  **for (List<String> productElement : cartesianProductList) {**  **for (String s : productElement) {**  **tempString.append(s);**  **tempString.append(ReportEngineConstants.STRING\_SEPERATOR\_RE);**  **}**  **downSection.add(tempString.toString().substring(0, tempString.toString().length() - 2));**  **tempString = new StringBuilder();**  **tempString.append(ReportEngineConstants.EMPTY);**  **}**  **}**  **return downSection;**  **}**  **/\*\***  **\* Method Name : cartesianProductColumnList Description : It will create the**  **\* cartesian product of database column list Creation Date : Mar 9, 2018 ,**  **\* 3:42:47 PM Created By : BV115775**  **\*/**  **public List<String> dbColumnCartesianProduct(List<Set<String>> keySet,**  **List<List<DataElementsWithColumns>> allReportParameter) {**  **List<String> columnList = new ArrayList<>();**  **int demoGroupCount = 0;**  **String demoGroupData = null;**  **String demo = RPT\_PROPERTY\_DEMOGRP;**  **List<String> keySetData = keySetList(keySet);**  **List<DataElementsWithColumns> allReportParameterData = combiningDataElementForReportLayout(allReportParameter);**  **for (DataElementsWithColumns dataElementsWithColumns : allReportParameterData) {**  **for (String keySetElementName : keySetData) {**  **if (keySetElementName.equals(dataElementsWithColumns.getDataElementId())) {**  **if (demo.equals(dataElementsWithColumns.getDataElementId())) {**  **demoGroupCount++;**  **demoGroupData = dataElementsWithColumns.getDbColumns();**  **} else {**  **columnList.add(dataElementsWithColumns.getDbColumns());**  **}**  **}**  **}**  **}**  **if (demoGroupCount == 1) {**  **columnList.add(demoGroupData);**  **}**  **return columnList;**  **}**  **public List<String> keySetList(List<Set<String>> keySet) {**  **List<String> keySetData = new LinkedList<>();**  **for (Set<String> keySetList : keySet) {**  **for (String keySetElementName : keySetList) {**  **keySetData.add(keySetElementName);**  **}**  **}**  **return keySetData;**  **}**  **}** |

**Component test:**

|  |
| --- |
| **import java.util.ArrayList;**  **import java.util.Arrays;**  **import java.util.HashMap;**  **import java.util.HashSet;**  **import java.util.List;**  **import java.util.Map;**  **import java.util.Set;**  **import org.junit.Test;**  **import org.springframework.boot.test.context.SpringBootTest;**  **import com.nbcu.tvrocs.reportengine.LMWReportEngineApplication;**  **import com.nbcu.tvrocs.reportengine.model.DataElementsWithColumns;**  **import com.nbcu.tvrocs.reportengine.utils.ReportEngineConstants;**  **@SpringBootTest(classes = LMWReportEngineApplication.class)**  **public class CartesianProductTest {**  **@Test**  **public void cartesianProductColumnListTest() {**  **Set<String> testValue = new HashSet<>();**  **testValue.add("Network Code");**  **List<Set<String>> keySet = new ArrayList<>();**  **keySet.add(testValue);**  **DataElementsWithColumns dataElementsWithColumns = new DataElementsWithColumns("String", "Network Code");**  **dataElementsWithColumns.setDbColumns("C8037\_NETWORK\_CD");**  **DataElementsWithColumns dataElementsWithColumn = new DataElementsWithColumns("String", "Broadcast Date");**  **dataElementsWithColumn.setDbColumns("C8020\_CALENDAR\_DT");**  **List<DataElementsWithColumns> list = new ArrayList<>();**  **list.add(dataElementsWithColumns);**  **list.add(dataElementsWithColumn);**  **List<List<DataElementsWithColumns>> allReportParameter = new ArrayList<>();**  **allReportParameter.add(list);**  **CartesianProduct cartesianProduct = new CartesianProduct();**  **cartesianProduct.dbColumnCartesianProduct(keySet, allReportParameter);**  **}**  **@Test**  **public void cartesianProductColumnListWithDemoTest() {**  **Set<String> testValue = new HashSet<>();**  **testValue.add("RPT\_PROPERTY\_NETWORK");**  **List<Set<String>> keySet = new ArrayList<>();**  **keySet.add(testValue);**  **testValue.add("RPT\_PROPERTY\_DEMOGRP");**  **keySet.add(testValue);**  **DataElementsWithColumns dataElementsWithColumns = new DataElementsWithColumns("String", "RPT\_PROPERTY\_NETWORK");**  **dataElementsWithColumns.setDbColumns("C8037\_NETWORK\_CD");**  **DataElementsWithColumns dataElementsWithColumn = new DataElementsWithColumns("String", "RPT\_PROPERTY\_DEMOGRP");**  **dataElementsWithColumn.setDbColumns(" ");**  **List<DataElementsWithColumns> list = new ArrayList<>();**  **list.add(dataElementsWithColumns);**  **list.add(dataElementsWithColumn);**  **List<List<DataElementsWithColumns>> allReportParameter = new ArrayList<>();**  **allReportParameter.add(list);**  **CartesianProduct cartesianProduct = new CartesianProduct();**  **cartesianProduct.dbColumnCartesianProduct(keySet, allReportParameter);**  **}**  **@Test**  **public void createAcrossMetricCartesianProductTest() {**  **List<String> Across = Arrays.asList("HH");**  **List<String> metrics = Arrays.asList("TV(000)");**  **Map<String, List<String>> uniqueAcrossElementsData = new HashMap<>();**  **Map<String, List<String>> uniqueMetricElementsData = new HashMap<>();**  **uniqueAcrossElementsData.put("Demo Group", Across);**  **uniqueMetricElementsData.put("Metrics", metrics);**  **CartesianProduct cartesianProduct = new CartesianProduct();**  **Map<String, Map<String, List<String>>> unqiueDataElementList = new HashMap<>();**  **unqiueDataElementList.put(ReportEngineConstants.ACROSS, uniqueAcrossElementsData);**  **unqiueDataElementList.put(ReportEngineConstants.METRICS, uniqueMetricElementsData);**  **cartesianProduct.aCrossMetricCartesianProduct(unqiueDataElementList);**  **}**  **@Test**  **public void createAcrossMetricsCartesianByUniqueDataTest() {**  **Map<String, Map<String, List<String>>> unqiueDataElementList = new HashMap<>();**  **Map<String, List<String>> temp = new HashMap();**  **temp.put("temp", Arrays.asList("Ny,HH", "La,HH"));**  **unqiueDataElementList.put(ReportEngineConstants.ACROSS, temp);**  **unqiueDataElementList.put(ReportEngineConstants.METRICS, temp);**  **CartesianProduct cartesianProduct = new CartesianProduct();**  **cartesianProduct.aCrossMetricsCartesianByUniqueData(unqiueDataElementList);**  **}**  **@Test**  **public void createCartesianProductTest() {**  **List<String> arrayList = Arrays.asList("NBC", "CBS");**  **Map<String, List<String>> uniqueElementsData = new HashMap<>();**  **uniqueElementsData.put("Network Code", arrayList);**  **CartesianProduct cartesianProduct = new CartesianProduct();**  **cartesianProduct.createCartesianProduct(uniqueElementsData);**  **}**  **}** |