**Service Unit Testing**

Contents

[1. Service 1 2](#_Toc521501603)

[2. Service 2 16](#_Toc521501604)

[3. Service 3 18](#_Toc521501605)

1. **Service 1**

Service 1 Test:

Service class

|  |
| --- |
| import java.util.ArrayList;  import java.util.Arrays;  import java.util.List;  import java.util.Optional;  import org.apache.commons.collections4.CollectionUtils;  import org.apache.commons.lang.StringUtils;  import org.slf4j.Logger;  import org.slf4j.LoggerFactory;  import org.springframework.beans.factory.annotation.Autowired;  import org.springframework.stereotype.Service;  import org.springframework.transaction.annotation.Transactional;  import com.aspose.cells.Workbook;  import com.google.gson.Gson;  import com.nbcu.tvrocs.model.ui.Report;  import com.nbcu.tvrocs.model.ui.RequestParam;  import com.nbcu.tvrocs.reportconsolidator.aws.RCFileUploadDownload;  import com.nbcu.tvrocs.reportconsolidator.entity.ExecutionStatus;  import com.nbcu.tvrocs.reportconsolidator.entity.RequestStatusEntity;  import com.nbcu.tvrocs.reportconsolidator.model.RCRequest;  import com.nbcu.tvrocs.reportconsolidator.producer.RCProducer;  import com.nbcu.tvrocs.reportconsolidator.repo.ExecutionStatusRepo;  import com.nbcu.tvrocs.reportconsolidator.repo.RequestStatusRepo;  import com.nbcu.tvrocs.reportconsolidator.utils.LMWReportFormat;  import com.nbcu.tvrocs.reportconsolidator.utils.RCConstants;  import com.nbcu.tvrocs.reportconsolidator.utils.RCUtils;  /\*\*  \*  \* This class will be called from RCConsumer, once the message is received from  \* the listener.  \*  \* @author 206572421  \*  \*/  @Service  public class RCService {  static final Logger logger = LoggerFactory.getLogger(RCService.class);  private static final List<String> reportStatus = new ArrayList<>(  Arrays.asList("NODATAFOUND", "COMPLETED", "COMPLETED\_ERROR", "COMPLETED\_MAX\_ROW\_COUNT\_ERROR"));  @Autowired  private RCFileUploadDownload rcUploadDownload;  @Autowired  private RCProducer producer;  @Autowired  ExecutionStatusRepo executionStatusRepo;  @Autowired  RequestStatusRepo reqStatusRepo;  /\*\*  \* This method will process the execution-status with initial validation of the  \* request.  \*  \* @param rcRequest  \*/  public void processExecutionIdStatus(RCRequest rcRequest) {  if (null != rcRequest && StringUtils.isNotBlank(rcRequest.getExecutionId())) {  long start = System.currentTimeMillis();  executeStatusProcessor(rcRequest);  logger.info("Time taken to process the request -> {} milliseconds", System.currentTimeMillis() - start);  }  }  @Transactional  public int updateRequestStatus(String requestId) {  return reqStatusRepo.updateRequestStatus(requestId);  }  /\*\*  \* @param rcRequest  \* @param workBook  \* @param executionStatus  \* @param requestId  \* @param countOfRequest  \* @param requestParam  \* @throws Exception  \*/  private void checkReportStatus(RCRequest rcRequest, List<ExecutionStatus> executionStatus, String requestId,  int countOfRequest, RequestParam requestParam, String executionId) throws Exception {  int noDataFoundCount = 0;  int errorCount = 0;  int maxRowReachedCount = 0;  int completedCount = 0;  for (ExecutionStatus executionStatusLst : executionStatus) {  switch(executionStatusLst.getReportStatus())  {  case RCConstants.COMPLETED\_ERROR:  errorCount = errorCount + 1;  break;  case RCConstants.NO\_DATA\_FOUND:  noDataFoundCount = noDataFoundCount + 1;  break;  case RCConstants.MAX\_ROW\_COUNT\_REACHED\_ERROR:  maxRowReachedCount = maxRowReachedCount + 1;  break;  case RCConstants.STATUS\_COMPLETED:  if(StringUtils.isNotBlank(executionStatusLst.getReportOutputUrl()))  {  completedCount = completedCount + 1;  }  break;  default:  break;  }  }  // Calculate the finalCount based on the each status  int finalCount = errorCount + noDataFoundCount + maxRowReachedCount + completedCount;  // Proceed with consolidation, only if the totalcount & finalcount matches.  if (countOfRequest == finalCount) {  // Update LMW\_UI\_REQUEST\_STATUS Table  int updateRowCount = updateRequestStatus(RCUtils.getRequestId(executionId));  // Proceed with consolidation, only if update returns count. This check is done,  // to avoid multiple instances working.  if (updateRowCount > 0) {  if (noDataFoundCount == countOfRequest) {  // Send as NO\_DATA\_FOUND to SS.  producer.publishToStatus(null, requestId, rcRequest.getTemplateName(), null,  RCConstants.NO\_DATA\_FOUND);  } else if (maxRowReachedCount == countOfRequest) {  // Send as MAX\_ROW\_REACHED to SS.  producer.publishToStatus(null, requestId, rcRequest.getTemplateName(), null,  RCConstants.MAX\_ROW\_COUNT\_REACHED\_ERROR);  } else if (errorCount == countOfRequest) {  // Send as FAILED to SS.  producer.publishToStatus(null, requestId, rcRequest.getTemplateName(), null,  RCConstants.STATUS\_FAILED);  } else {  proceedWithConsolidation(rcRequest, executionStatus, requestId, requestParam);  }  }  }  }  /\*\*  \* This method will first check;  \*  \* IF count from executionId, is =1, THEN It will proceed with excel creation in  \* S3 and publish status to SS accordingly. IF totalcount matches the  \* NODATAFOUND THEN send status to SS as NODATAFOUND or totalcount matches the  \* COMPLETED ERROR THEN send status to SS as FAILED ELSE Will call  \* checkMapAndProceed method to update the map and proceed accordingly.  \*  \* @param rcRequest  \*/  private void executeStatusProcessor(RCRequest rcRequest) {  String executionId = rcRequest.getExecutionId();  try {  // Retrieve the details from the DB for the executionId from  // LMW\_EXECUTION\_STATUS.  List<ExecutionStatus> executionStatus = executionStatusRepo  .findByRequestIdAndReportStatusIn(RCUtils.getRequestId(executionId), reportStatus);  // Get the Request Id.  String requestId = executionStatus.get(0).getRequestId();  // Get the count of the sub-reports from requestId.  int countOfRequest = Integer.parseInt(RCUtils.getCountFromRequest(requestId));  // Check if total count matches from the DB.  if (countOfRequest == executionStatus.size()) {  retrieveRequestParamAndProceed(rcRequest, executionId, executionStatus, requestId, countOfRequest);  }  } catch (Exception e) {  logger.error("MethodName :: executionStatusProcessor , RequestId :: NA, ExecutionId :: {}, Error Message :: {}",  rcRequest.getExecutionId(), RCUtils.convertStackTraceToString(e));  producer.publishToStatus(executionId, null, rcRequest.getTemplateName(), null, RCConstants.STATUS\_FAILED);  }  }  /\*\*  \* @param rcRequest  \* @param workBook  \* @param executionStatus  \* @param requestId  \* @param requestParam  \* @throws Exception  \*/  private void proceedWithConsolidation(RCRequest rcRequest, List<ExecutionStatus> executionStatus, String requestId,  RequestParam requestParam) throws Exception {  // Create a workBook object  Workbook workBook = new Workbook();  // Clear any sheets, and keep it fresh.  workBook.getWorksheets().clear();  List<Report> reports = requestParam.getReports();  List<String> reportName = new ArrayList<>();  reports.forEach(report -> reportName.add(report.getReportName()));  for (String report : reportName) {  for (ExecutionStatus execution : executionStatus) {  if (execution.getReportName().equals(report)) {  try {  rcUploadDownload.callS3ForReport(execution.getReportStatus(), execution.getExecutionId(),  execution.getReportName(), workBook);  } catch (Exception e) {  logger.error("MethodName :: proceedWithConsolidation , RequestId :: NA, ExecutionId :: {}, Error Message :: {}",  rcRequest.getExecutionId(), RCUtils.convertStackTraceToString(e));  throw e;  }  }  }  }  // Call to save and upload the Excel book.  String fileName = saveUploadFile(requestId, workBook, rcRequest.getTemplateName());  // publish status to SS.  publishStatus(rcRequest.getTemplateName(), requestId, fileName, RCConstants.STATUS\_SUCCESS);  }  /\*\*  \* This method will call producer to publish the status to SS in case  \*  \* @param rcRequest  \* @param fileName  \*/  private void publishStatus(String templateName, String requestId, String fileName, String status) {  // Frame reportOutputURL  String outPutUrl = RCConstants.S3\_URL.concat(fileName);  // Publish the status to SS  producer.publishToStatus(null, requestId, templateName, outPutUrl, status);  }  private void retrieveRequestParamAndProceed(RCRequest rcRequest, String executionId,  List<ExecutionStatus> executionStatus, String requestId, int countOfRequest) throws Exception {  // Retrieve the Request Parameters for the request id to process the reportName  // to reflect in excel tab.  Optional<RequestStatusEntity> reqStatus = reqStatusRepo.findById(requestId);  // Get the RequestParam.  RequestParam requestParam = new Gson()  .fromJson(reqStatus.isPresent() ? reqStatus.get().getRequestParameters() : null, RequestParam.class);  if (null != requestParam && CollectionUtils.isNotEmpty(requestParam.getReports())) {  checkReportStatus(rcRequest, executionStatus, requestId, countOfRequest, requestParam, executionId);  }  }  /\*\*  \* This method will save the workBook locally & call upload file method.  \*  \* @param requestId  \* @param workBook  \* @param requestStatus  \* @throws Exception  \*/  private String saveUploadFile(String requestId, Workbook workBook, String templateName) throws Exception {  // Generate the filename.  String fileName = requestId.concat(RCConstants.HYPHEN).concat(templateName)  .concat(LMWReportFormat.EXCEL.getValue());  // Save the workbook locally within the project  workBook.save(fileName);  logger.info("MethodName :: saveUploadFile , RequestId :: NA, ExecutionId :: {}, Message ::Worbook saved locally with fileName :{}. Total worksheet counts: {}",  requestId, fileName, workBook.getWorksheets().getCount());  // Call to upload the file to S3.  rcUploadDownload.uploadFileToS3(requestId, fileName, workBook);  return fileName;  }  } |

**Service Test:**

|  |
| --- |
| import static org.mockito.Mockito.when;  import java.io.BufferedReader;  import java.io.File;  import java.io.FileReader;  import java.io.IOException;  import java.sql.Timestamp;  import java.util.ArrayList;  import java.util.Arrays;  import java.util.List;  import java.util.Optional;  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.boot.test.context.SpringBootTest;  import com.amazonaws.services.s3.AmazonS3;  import com.amazonaws.services.s3.model.GetObjectRequest;  import com.amazonaws.services.s3.model.S3Object;  import com.aspose.cells.Workbook;  import com.nbcu.tvrocs.reportconsolidator.LMWReportConsolidatorApplication;  import com.nbcu.tvrocs.reportconsolidator.aws.RCFileUploadDownload;  import com.nbcu.tvrocs.reportconsolidator.entity.ExecutionStatus;  import com.nbcu.tvrocs.reportconsolidator.entity.RequestStatusEntity;  import com.nbcu.tvrocs.reportconsolidator.model.RCRequest;  import com.nbcu.tvrocs.reportconsolidator.producer.RCProducer;  import com.nbcu.tvrocs.reportconsolidator.repo.ExecutionStatusRepo;  import com.nbcu.tvrocs.reportconsolidator.repo.RequestStatusRepo;  import com.nbcu.tvrocs.reportconsolidator.utils.RCConstants;  @SpringBootTest(classes = LMWReportConsolidatorApplication.class)  public class RCServiceTest {  private static final List<String> reportStatus = new ArrayList<>(  Arrays.asList("NODATAFOUND", "COMPLETED", "COMPLETED\_ERROR", "COMPLETED\_MAX\_ROW\_COUNT\_ERROR"));  public static String readFile() {  BufferedReader reader = null;  try {  File inputXmlFile = new File(new RCServiceTest().getClass().getResource("/inputJson.txt").getFile());  reader = new BufferedReader(new FileReader(inputXmlFile));  String line = null;  StringBuilder stringBuilder = new StringBuilder();  while ((line = reader.readLine()) != null) {  stringBuilder.append(line);  }  return stringBuilder.toString();  } catch (Exception e) {  return null;  } finally {  if (reader != null) {  try {  reader.close();  } catch (IOException e) {  }  }  }  }  @Mock  private RCProducer producer;  @Mock  ExecutionStatusRepo executionStatusRepo;  @Mock  RequestStatusRepo reqStatusRepo;  @InjectMocks  RCService rcService;  @Mock  RCFileUploadDownload rcUploadDownload;  @Mock  AmazonS3 amazonS3;    List<ExecutionStatus> executionStatusList;  @Test  public void processExecutionIdStatus() {  RCRequest request = new RCRequest();  request.setExecutionId("");  rcService.processExecutionIdStatus(request);  }  @Test  public void processExecutionIdStatus1() throws Exception {  RCRequest request = new RCRequest();  request.setExecutionId("206572421-71d67f0a-7071-4df5-8ec9-402f4e9de40c-03-01-Report 1");  request.setTemplateName("test");  RequestStatusEntity requestStatusObj = new RequestStatusEntity();  requestStatusObj.setRequestParameters(readFile());  Optional<RequestStatusEntity> requestStatus = Optional.of((RequestStatusEntity) requestStatusObj);  List<ExecutionStatus> executionStatus = new ArrayList<>();  ExecutionStatus executionStatusEntity = new ExecutionStatus();  executionStatusEntity.setDataEngineSql("SampleSql");  executionStatusEntity.setRequestId("206572421-71d67f0a-7071-4df5-8ec9-402f4e9de40c-03");  executionStatusEntity.setExecutionId("206572421-71d67f0a-7071-4df5-8ec9-402f4e9de40c-03-01-Report 1");  executionStatusEntity.setReportEndDateTime(new Timestamp(System.currentTimeMillis()));  executionStatusEntity.setReportEngineSql("ReportEngineSql");  executionStatusEntity.setReportId("ReportId");  executionStatusEntity.setReportMessage("Completed");  executionStatusEntity.setReportName("Report 1");  executionStatusEntity.setReportOutputUrl(  "https://lmw-poc.grid.tfayd.com:443/206572421-71d67f0a-7071-4df5-8ec9-402f4e9de40c-03-01-Report 1.xlsx");  executionStatusEntity.setReportParameters("ReportParameters");  executionStatusEntity.setReportStartDateTime(new Timestamp(System.currentTimeMillis()));  executionStatusEntity.setReportStatus("COMPLETED");  executionStatusEntity.setUpdatedBy("Re");  ExecutionStatus executionStatusEntity1 = new ExecutionStatus();  executionStatusEntity1.setDataEngineSql("SampleSql");  executionStatusEntity1.setRequestId("206572421-71d67f0a-7071-4df5-8ec9-402f4e9de40c-03");  executionStatusEntity1.setExecutionId("206572421-71d67f0a-7071-4df5-8ec9-402f4e9de40c-03-02-Report 2");  executionStatusEntity1.setReportEndDateTime(new Timestamp(System.currentTimeMillis()));  executionStatusEntity1.setReportEngineSql("ReportEngineSql");  executionStatusEntity1.setReportId("ReportId");  executionStatusEntity1.setReportMessage("Completed");  executionStatusEntity1.setReportName("Report 2");  executionStatusEntity1.setReportOutputUrl(  "https://lmw-poc.grid.tfayd.com:443/206572421-71d67f0a-7071-4df5-8ec9-402f4e9de40c-03-02-Report 2.xlsx");  executionStatusEntity1.setReportParameters("ReportParameters");  executionStatusEntity1.setReportStartDateTime(new Timestamp(System.currentTimeMillis()));  executionStatusEntity1.setReportStatus("COMPLETED");  executionStatusEntity1.setUpdatedBy("Re");  ExecutionStatus executionStatusEntity2 = new ExecutionStatus();  executionStatusEntity2.setDataEngineSql("SampleSql");  executionStatusEntity2.setRequestId("206572421-71d67f0a-7071-4df5-8ec9-402f4e9de40c-03");  executionStatusEntity2.setExecutionId("206572421-71d67f0a-7071-4df5-8ec9-402f4e9de40c-03-03-Report 3");  executionStatusEntity2.setReportEndDateTime(new Timestamp(System.currentTimeMillis()));  executionStatusEntity2.setReportEngineSql("ReportEngineSql");  executionStatusEntity2.setReportId("ReportId");  executionStatusEntity2.setReportMessage("Completed");  executionStatusEntity2.setReportName("Report 3");  executionStatusEntity2.setReportOutputUrl(  "https://lmw-poc.grid.tfayd.com:443/206572421-71d67f0a-7071-4df5-8ec9-402f4e9de40c-03-03-Report 3.xlsx");  executionStatusEntity2.setReportParameters("ReportParameters");  executionStatusEntity2.setReportStartDateTime(new Timestamp(System.currentTimeMillis()));  executionStatusEntity2.setReportStatus("COMPLETED");  executionStatus.add(executionStatusEntity);  executionStatus.add(executionStatusEntity1);  executionStatus.add(executionStatusEntity2);  Mockito.when(executionStatusRepo  .findByRequestIdAndReportStatusIn("206572421-71d67f0a-7071-4df5-8ec9-402f4e9de40c-03", reportStatus))  .thenReturn(executionStatus);  Mockito.when(amazonS3.getObject(  new GetObjectRequest("lmw-poc", "206572421-71d67f0a-7071-4df5-8ec9-402f4e9de40c-03-03-Report 3.xlsx")))  .thenReturn(new S3Object());  Mockito.when(reqStatusRepo.updateRequestStatus("206572421-71d67f0a-7071-4df5-8ec9-402f4e9de40c-03"))  .thenReturn(1);  Mockito.when(reqStatusRepo.findById("206572421-71d67f0a-7071-4df5-8ec9-402f4e9de40c-03"))  .thenReturn(requestStatus);  Mockito.doNothing().when(producer).publishToStatus(null, "206572421-71d67f0a-7071-4df5-8ec9-402f4e9de40c-03",  "test", "", RCConstants.STATUS\_SUCCESS);  Mockito.doNothing().when(rcUploadDownload).callS3ForReport("COMPLETED",  "206572421-71d67f0a-7071-4df5-8ec9-402f4e9de40c-03-Report 3",  "206572421-71d67f0a-7071-4df5-8ec9-402f4e9de40c-03", new Workbook());  rcService.processExecutionIdStatus(request);  }  @Test  public void processExecutionIdStatus2() throws Exception {  RCRequest request = new RCRequest();  request.setExecutionId("206572421-71d67f0a-7071-4df5-8ec9-402f4e9de40c-03-01-Report 1");  request.setTemplateName("test");  RequestStatusEntity requestStatusObj = new RequestStatusEntity();  requestStatusObj.setRequestParameters(readFile());  Optional<RequestStatusEntity> requestStatus = Optional.of((RequestStatusEntity) requestStatusObj);  List<ExecutionStatus> executionStatus = new ArrayList<>();  ExecutionStatus executionStatusEntity = new ExecutionStatus();  executionStatusEntity.setDataEngineSql("SampleSql");  executionStatusEntity.setRequestId("206572421-71d67f0a-7071-4df5-8ec9-402f4e9de40c-03");  executionStatusEntity.setExecutionId("206572421-71d67f0a-7071-4df5-8ec9-402f4e9de40c-03-01-Report 1");  executionStatusEntity.setReportEndDateTime(new Timestamp(System.currentTimeMillis()));  executionStatusEntity.setReportEngineSql("ReportEngineSql");  executionStatusEntity.setReportId("ReportId");  executionStatusEntity.setReportMessage("COMPLETED\_ERROR");  executionStatusEntity.setReportName("Report 1");  executionStatusEntity.setReportOutputUrl(  "https://lmw-poc.grid.tfayd.com:443/206572421-71d67f0a-7071-4df5-8ec9-402f4e9de40c-03-01-Report 1.xlsx");  executionStatusEntity.setReportParameters("ReportParameters");  executionStatusEntity.setReportStartDateTime(new Timestamp(System.currentTimeMillis()));  executionStatusEntity.setReportStatus("COMPLETED\_ERROR");  executionStatusEntity.setUpdatedBy("Re");  ExecutionStatus executionStatusEntity1 = new ExecutionStatus();  executionStatusEntity1.setDataEngineSql("SampleSql");  executionStatusEntity1.setRequestId("206572421-71d67f0a-7071-4df5-8ec9-402f4e9de40c-03");  executionStatusEntity1.setExecutionId("206572421-71d67f0a-7071-4df5-8ec9-402f4e9de40c-03-02-Report 2");  executionStatusEntity1.setReportEndDateTime(new Timestamp(System.currentTimeMillis()));  executionStatusEntity1.setReportEngineSql("ReportEngineSql");  executionStatusEntity1.setReportId("ReportId");  executionStatusEntity1.setReportMessage("COMPLETED\_ERROR");  executionStatusEntity1.setReportName("Report 2");  executionStatusEntity1.setReportOutputUrl(  "https://lmw-poc.grid.tfayd.com:443/206572421-71d67f0a-7071-4df5-8ec9-402f4e9de40c-03-02-Report 2.xlsx");  executionStatusEntity1.setReportParameters("ReportParameters");  executionStatusEntity1.setReportStartDateTime(new Timestamp(System.currentTimeMillis()));  executionStatusEntity1.setReportStatus("COMPLETED\_ERROR");  executionStatusEntity1.setUpdatedBy("Re");  ExecutionStatus executionStatusEntity2 = new ExecutionStatus();  executionStatusEntity2.setDataEngineSql("SampleSql");  executionStatusEntity2.setRequestId("206572421-71d67f0a-7071-4df5-8ec9-402f4e9de40c-03");  executionStatusEntity2.setExecutionId("206572421-71d67f0a-7071-4df5-8ec9-402f4e9de40c-03-03-Report 3");  executionStatusEntity2.setReportEndDateTime(new Timestamp(System.currentTimeMillis()));  executionStatusEntity2.setReportEngineSql("ReportEngineSql");  executionStatusEntity2.setReportId("ReportId");  executionStatusEntity2.setReportMessage("COMPLETED\_ERROR");  executionStatusEntity2.setReportName("Report 3");  executionStatusEntity2.setReportOutputUrl(  "https://lmw-poc.grid.tfayd.com:443/206572421-71d67f0a-7071-4df5-8ec9-402f4e9de40c-03-03-Report 3.xlsx");  executionStatusEntity2.setReportParameters("ReportParameters");  executionStatusEntity2.setReportStartDateTime(new Timestamp(System.currentTimeMillis()));  executionStatusEntity2.setReportStatus("COMPLETED\_ERROR");  executionStatus.add(executionStatusEntity);  executionStatus.add(executionStatusEntity1);  executionStatus.add(executionStatusEntity2);  Mockito.when(executionStatusRepo  .findByRequestIdAndReportStatusIn("206572421-71d67f0a-7071-4df5-8ec9-402f4e9de40c-03", reportStatus))  .thenReturn(executionStatus);  Mockito.when(amazonS3.getObject(  new GetObjectRequest("lmw-poc", "206572421-71d67f0a-7071-4df5-8ec9-402f4e9de40c-03-03-Report 3.xlsx")))  .thenReturn(new S3Object());  Mockito.when(reqStatusRepo.updateRequestStatus("206572421-71d67f0a-7071-4df5-8ec9-402f4e9de40c-03"))  .thenReturn(1);  Mockito.when(reqStatusRepo.findById("206572421-71d67f0a-7071-4df5-8ec9-402f4e9de40c-03"))  .thenReturn(requestStatus);  Mockito.doNothing().when(producer).publishToStatus(null, "206572421-71d67f0a-7071-4df5-8ec9-402f4e9de40c-03",  "test", "", RCConstants.STATUS\_SUCCESS);  Mockito.doNothing().when(rcUploadDownload).callS3ForReport("COMPLETED",  "206572421-71d67f0a-7071-4df5-8ec9-402f4e9de40c-03-Report 3",  "206572421-71d67f0a-7071-4df5-8ec9-402f4e9de40c-03", new Workbook());  rcService.processExecutionIdStatus(request);  }  @Test  public void processExecutionIdStatus3() throws Exception {  RCRequest request = new RCRequest();  request.setExecutionId("206572421-71d67f0a-7071-4df5-8ec9-402f4e9de40c-03-01-Report 1");  request.setTemplateName("test");  RequestStatusEntity requestStatusObj = new RequestStatusEntity();  requestStatusObj.setRequestParameters(readFile());  Optional<RequestStatusEntity> requestStatus = Optional.of((RequestStatusEntity) requestStatusObj);  List<ExecutionStatus> executionStatus = new ArrayList<>();  ExecutionStatus executionStatusEntity = new ExecutionStatus();  executionStatusEntity.setDataEngineSql("SampleSql");  executionStatusEntity.setRequestId("206572421-71d67f0a-7071-4df5-8ec9-402f4e9de40c-03");  executionStatusEntity.setExecutionId("206572421-71d67f0a-7071-4df5-8ec9-402f4e9de40c-03-01-Report 1");  executionStatusEntity.setReportEndDateTime(new Timestamp(System.currentTimeMillis()));  executionStatusEntity.setReportEngineSql("ReportEngineSql");  executionStatusEntity.setReportId("ReportId");  executionStatusEntity.setReportStatus("NODATAFOUND");  executionStatusEntity.setReportName("Report 1");  executionStatusEntity.setReportOutputUrl(  "https://lmw-poc.grid.tfayd.com:443/206572421-71d67f0a-7071-4df5-8ec9-402f4e9de40c-03-01-Report 1.xlsx");  executionStatusEntity.setReportParameters("ReportParameters");  executionStatusEntity.setReportStartDateTime(new Timestamp(System.currentTimeMillis()));  executionStatusEntity.setReportMessage("COMPLETED\_ERROR");  executionStatusEntity.setUpdatedBy("Re");  ExecutionStatus executionStatusEntity1 = new ExecutionStatus();  executionStatusEntity1.setDataEngineSql("SampleSql");  executionStatusEntity1.setRequestId("206572421-71d67f0a-7071-4df5-8ec9-402f4e9de40c-03");  executionStatusEntity1.setExecutionId("206572421-71d67f0a-7071-4df5-8ec9-402f4e9de40c-03-02-Report 2");  executionStatusEntity1.setReportEndDateTime(new Timestamp(System.currentTimeMillis()));  executionStatusEntity1.setReportEngineSql("ReportEngineSql");  executionStatusEntity1.setReportId("ReportId");  executionStatusEntity1.setReportMessage("COMPLETED\_ERROR");  executionStatusEntity1.setReportName("Report 2");  executionStatusEntity1.setReportOutputUrl(  "https://lmw-poc.grid.tfayd.com:443/206572421-71d67f0a-7071-4df5-8ec9-402f4e9de40c-03-02-Report 2.xlsx");  executionStatusEntity1.setReportParameters("ReportParameters");  executionStatusEntity1.setReportStartDateTime(new Timestamp(System.currentTimeMillis()));  executionStatusEntity1.setReportStatus("NODATAFOUND");  executionStatusEntity1.setUpdatedBy("Re");  ExecutionStatus executionStatusEntity2 = new ExecutionStatus();  executionStatusEntity2.setDataEngineSql("SampleSql");  executionStatusEntity2.setRequestId("206572421-71d67f0a-7071-4df5-8ec9-402f4e9de40c-03");  executionStatusEntity2.setExecutionId("206572421-71d67f0a-7071-4df5-8ec9-402f4e9de40c-03-03-Report 3");  executionStatusEntity2.setReportEndDateTime(new Timestamp(System.currentTimeMillis()));  executionStatusEntity2.setReportEngineSql("ReportEngineSql");  executionStatusEntity2.setReportId("ReportId");  executionStatusEntity2.setReportMessage("NODATAFOUND");  executionStatusEntity2.setReportName("Report 3");  executionStatusEntity2.setReportOutputUrl(  "https://lmw-poc.grid.tfayd.com:443/206572421-71d67f0a-7071-4df5-8ec9-402f4e9de40c-03-03-Report 3.xlsx");  executionStatusEntity2.setReportParameters("ReportParameters");  executionStatusEntity2.setReportStartDateTime(new Timestamp(System.currentTimeMillis()));  executionStatusEntity2.setReportStatus("NODATAFOUND");  executionStatus.add(executionStatusEntity);  executionStatus.add(executionStatusEntity1);  executionStatus.add(executionStatusEntity2);  Mockito.when(executionStatusRepo  .findByRequestIdAndReportStatusIn("206572421-71d67f0a-7071-4df5-8ec9-402f4e9de40c-03", reportStatus))  .thenReturn(executionStatus);  Mockito.when(amazonS3.getObject(  new GetObjectRequest("lmw-poc", "206572421-71d67f0a-7071-4df5-8ec9-402f4e9de40c-03-03-Report 3.xlsx")))  .thenReturn(new S3Object());  Mockito.when(reqStatusRepo.updateRequestStatus("206572421-71d67f0a-7071-4df5-8ec9-402f4e9de40c-03"))  .thenReturn(1);  Mockito.when(reqStatusRepo.findById("206572421-71d67f0a-7071-4df5-8ec9-402f4e9de40c-03"))  .thenReturn(requestStatus);  Mockito.doNothing().when(producer).publishToStatus(null, "206572421-71d67f0a-7071-4df5-8ec9-402f4e9de40c-03",  "test", "", RCConstants.STATUS\_SUCCESS);  Mockito.doNothing().when(rcUploadDownload).callS3ForReport("COMPLETED",  "206572421-71d67f0a-7071-4df5-8ec9-402f4e9de40c-03-Report 3",  "206572421-71d67f0a-7071-4df5-8ec9-402f4e9de40c-03", new Workbook());  rcService.processExecutionIdStatus(request);  }  @Test  public void processExecutionIdStatus4() throws Exception {  RCRequest request = new RCRequest();  request.setExecutionId("206572421-71d67f0a-7071-4df5-8ec9-402f4e9de40c-03-01-Report 1");  request.setTemplateName("test");  when(executionStatusRepo.findByRequestIdAndReportStatusIn("206572421-71d67f0a-7071-4df5-8ec9-402f4e9de40c-03", reportStatus)).thenReturn(executionStatusList);  rcService.processExecutionIdStatus(request);  }  @Before  public void setUp() throws Exception {  MockitoAnnotations.initMocks(this);    executionStatusList = new ArrayList<>();  ExecutionStatus executionStatus = new ExecutionStatus();  executionStatus.setRequestId("206572421-71d67f0a-7071-4df5-8ec9-402f4e9de40c-03");  executionStatusList.add(executionStatus);  }  } |

1. **Service 2**

**Service class**

|  |
| --- |
| import org.slf4j.Logger;  import org.slf4j.LoggerFactory;  import org.springframework.amqp.rabbit.annotation.RabbitListener;  import org.springframework.beans.factory.annotation.Autowired;  import org.springframework.stereotype.Service;  import com.nbcu.tvrocs.reportconsolidator.model.RCRequest;  import com.nbcu.tvrocs.reportconsolidator.service.RCService;  import com.nbcu.tvrocs.reportconsolidator.utils.RCConstants;  /\*\*  \* This is a RabbitMQ Listener class for the Q5.  \*  \* @author 206572421  \*  \*/  @Service  public class RCConsumer {  static final Logger logger = LoggerFactory.getLogger(RCConsumer.class);  @Autowired  private RCService rcService;  /\*\*  \* This method will receive the message from Q5 where RE/SS puts the status of  \* the execution.  \* @param rcRequest  \* @throws InterruptedException  \*/  @RabbitListener(queues = RCConstants.RC\_QUEUE, containerFactory = "jsaFactory")  public void receiveMessage(RCRequest rcRequest) {  logger.info("MethodName :: receiveMessage , RequestId :: NA, ExecutionId :: {}, Message :: Message Recieved {}",  rcRequest.getExecutionId(), rcRequest);  rcService.processExecutionIdStatus(rcRequest);  }  } |

**Service Test:**

|  |
| --- |
| import org.junit.Before;  import org.junit.Test;  import org.mockito.InjectMocks;  import org.mockito.Mock;  import org.mockito.MockitoAnnotations;  import com.nbcu.tvrocs.reportconsolidator.model.RCRequest;  import com.nbcu.tvrocs.reportconsolidator.service.RCService;  public class RCConsumerTest {  @InjectMocks  private RCConsumer rcConsumer;    @Mock  private RCService rcService;    private RCRequest rcRequest;    @Before  public void setUp()  {  MockitoAnnotations.initMocks(this);  rcRequest= new RCRequest();  rcRequest.setExecutionId("206567792-71d67f0a-7071-4df5-8ec9-402f4e9de40c-01-01-Report 1");  }    @Test  public void receiveStatusTest()  {  rcConsumer.receiveMessage(rcRequest);  }  } |

1. **Service 3**

Service class

|  |
| --- |
| import java.sql.Timestamp;  import org.apache.commons.lang.StringUtils;  import org.slf4j.Logger;  import org.slf4j.LoggerFactory;  import org.springframework.amqp.rabbit.core.RabbitTemplate;  import org.springframework.beans.factory.annotation.Autowired;  import org.springframework.stereotype.Service;  import com.nbcu.tvrocs.reportconsolidator.model.LMWRequestStatus;  import com.nbcu.tvrocs.reportconsolidator.model.LMWStatus;  import com.nbcu.tvrocs.reportconsolidator.utils.MQConstants;  import com.nbcu.tvrocs.reportconsolidator.utils.RCConstants;  import com.nbcu.tvrocs.reportconsolidator.utils.RCUtils;  /\*\*  \* Will publish to SS exchange.  \*  \* @author 206572421  \*  \*/  @Service  public class RCProducer {  static final Logger logger = LoggerFactory.getLogger(RCProducer.class);  @Autowired  private RabbitTemplate rabbitTemplate;  /\*\*  \* Will publish the status to SS service.  \* @param executionId  \* @param templateName  \* @param outputURL  \* @throws Exception  \*/  public void publishToStatus(String executionId, String requestId, String templateName, String outputURL,  String status) {  LMWStatus lmwstatus = new LMWStatus();  transformLMWStatus(executionId, requestId, outputURL, templateName, lmwstatus, status);  rabbitTemplate.convertAndSend(MQConstants.EXCHANGE\_STATUS\_SERVICE, MQConstants.KEY\_STATUS\_SERVICE, lmwstatus);  logger.info("MethodName :: publishToStatus, RequestId :: {} , ExecutionId :: NA , Message :: Sent to SS RabbitMQ, at time :: {}",  lmwstatus.getRequestId(), lmwstatus.getRequestStatus().getRequestEndDateTime());  }  /\*\*  \* Transform to LMWStatus object  \* @param requestId  \* @param outputURL  \* @param templateName  \* @param lmwstatus  \* @throws Exception  \*/  private void transformLMWStatus(String executionId, String requestId, String outputURL, String templateName,  LMWStatus lmwstatus, String status) {  LMWRequestStatus requestStatus = new LMWRequestStatus();  lmwstatus.setRequestId(StringUtils.isNotBlank(executionId) ? RCUtils.getRequestId(executionId) : requestId);  requestStatus.setRequestEndDateTime(new Timestamp(System.currentTimeMillis()));  lmwstatus.setSsoId(StringUtils.isNotBlank(executionId) ? executionId.substring(0, executionId.indexOf('-'))  : requestId.substring(0, requestId.indexOf('-')));  requestStatus.setTemplateName(templateName);  if (StringUtils.isNotBlank(outputURL)) {  requestStatus.setRequestOutputURL(outputURL);  requestStatus.setRequestMessage(RCConstants.MESSAGE\_SUCCESS);  requestStatus.setRequestStatus(RCConstants.STATUS\_SUCCESS);  } else if (RCConstants.NO\_DATA\_FOUND.equals(status)) {  requestStatus.setRequestMessage(RCConstants.MESSAGE\_NODATA);  requestStatus.setRequestStatus(RCConstants.NO\_DATA\_FOUND);  } else if (RCConstants.MAX\_ROW\_COUNT\_REACHED\_ERROR.equals(status)) {  requestStatus.setRequestMessage(RCConstants.MESSAGE\_MAX\_ROW\_COUNT\_REACHED\_ERROR);  requestStatus.setRequestStatus(RCConstants.MAX\_ROW\_COUNT\_REACHED\_ERROR);  } else {  requestStatus.setRequestMessage(RCConstants.MESSAGE\_FAILED);  requestStatus.setRequestStatus(RCConstants.STATUS\_FAILED);  }  lmwstatus.setRequestStatus(requestStatus);  }  } |

**Service class 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.MockitoAnnotations;  import org.springframework.amqp.rabbit.core.RabbitTemplate;  import com.nbcu.tvrocs.reportconsolidator.model.LMWStatus;  import com.nbcu.tvrocs.reportconsolidator.utils.MQConstants;  import com.nbcu.tvrocs.reportconsolidator.utils.RCConstants;  public class RCProducerTest {  @InjectMocks  RCProducer rcProducer;  @Mock  RabbitTemplate rabbitTemplate;  @Before  public void setUp() {  MockitoAnnotations.initMocks(this);  }  @Test  public void testPublishToStatusFailed() {  LMWStatus lmwstatus = new LMWStatus();  doNothing().when(rabbitTemplate).convertAndSend((MQConstants.EXCHANGE\_STATUS\_SERVICE),  MQConstants.KEY\_STATUS\_SERVICE, lmwstatus);  rcProducer.publishToStatus("206567792-71d67f0a-7071-4df5-8ec9-402f4e9de40c-01-01-Report 1", "requestId",  "templateName", "", "status");  }  @Test  public void testPublishToStatusMaxRow() {  LMWStatus lmwstatus = new LMWStatus();  doNothing().when(rabbitTemplate).convertAndSend((MQConstants.EXCHANGE\_STATUS\_SERVICE),  MQConstants.KEY\_STATUS\_SERVICE, lmwstatus);  rcProducer.publishToStatus("206567792-71d67f0a-7071-4df5-8ec9-402f4e9de40c-01-01-Report 1", "requestId",  "templateName", "", RCConstants.MAX\_ROW\_COUNT\_REACHED\_ERROR);  }  @Test  public void testPublishToStatusNoData() {  LMWStatus lmwstatus = new LMWStatus();  doNothing().when(rabbitTemplate).convertAndSend((MQConstants.EXCHANGE\_STATUS\_SERVICE),  MQConstants.KEY\_STATUS\_SERVICE, lmwstatus);  rcProducer.publishToStatus("206567792-71d67f0a-7071-4df5-8ec9-402f4e9de40c-01-01-Report 1", "requestId",  "templateName", "", RCConstants.NO\_DATA\_FOUND);  }  @Test  public void testPublishToStatusSuccess() {  LMWStatus lmwstatus = new LMWStatus();  doNothing().when(rabbitTemplate).convertAndSend((MQConstants.EXCHANGE\_STATUS\_SERVICE),  MQConstants.KEY\_STATUS\_SERVICE, lmwstatus);  rcProducer.publishToStatus("206567792-71d67f0a-7071-4df5-8ec9-402f4e9de40c-01-01-Report 1", "requestId",  "templateName", "outputURL", "status");  }  } |