Service Class-DocumentGeneratorEventStoreServiceImpl

package com.project.bos.dg.datastore.service.impl;  
  
import com.project.bos.dg.datastore.mapper.EventMapper;  
import com.project.bos.dg.datastore.model.entity.\*;  
import com.project.bos.dg.datastore.model.request.\*;  
import com.project.bos.dg.datastore.model.request.documentbydocumentIds.DocumentId;  
import com.project.bos.dg.datastore.model.request.documentbydocumentIds.DocumentIdsRequest;  
import com.project.bos.dg.datastore.model.response.EventDataResponse;  
import com.project.bos.dg.datastore.model.response.EventResponse;  
import com.project.bos.dg.datastore.model.response.EventStatusResponse;  
import com.project.bos.dg.datastore.model.response.common.DocumentResponse;  
import com.project.bos.dg.datastore.repository.\*;  
import jakarta.ws.rs.BadRequestException;  
import org.hibernate.query.sqm.mutation.internal.cte.AbstractCteMutationHandler;  
import org.junit.jupiter.api.Test;  
import org.junit.jupiter.api.function.Executable;  
import org.junit.runner.RunWith;  
import org.mockito.InjectMocks;  
import org.mockito.Mock;  
import org.mockito.junit.MockitoJUnitRunner;  
import org.springframework.http.HttpStatus;  
import org.springframework.http.HttpStatusCode;  
import org.springframework.web.client.HttpServerErrorException;  
  
import java.lang.reflect.InvocationTargetException;  
import java.lang.reflect.Method;  
import java.util.\*;  
  
  
import static org.junit.jupiter.api.Assertions.\*;  
import static org.mockito.Mockito.\*;  
  
@RunWith(MockitoJUnitRunner.class)  
class DocumentGeneratorEventStoreServiceImplTest {  
 @InjectMocks  
 private DocumentGeneratorEventStoreServiceImpl documentGeneratorEventStoreServiceImpl;  
  
 @Mock  
 private DocumentRepository documentRepository;  
 @Mock  
 private EventMapper eventMapper;  
 @Mock  
 private DmEventDocumentRepository dmEventDocumentRepository;  
 @Mock  
 private DmEventErrorRepository dmEventErrorRepository;  
 @Mock  
 private DmEventNotifyRepository dmEventNotifyRepository;  
 @Mock  
 private DmEventRepository dmEventRepository;  
  
  
 //Method 14  
 // Scenario 1 -Success Scenario  
 @Test  
 public void testFetchDocumentsByDocumentIds() {  
 //Arrange  
 DocumentIdsRequest documentIdsRequest = *mock*(DocumentIdsRequest.class);  
 List<DocumentId> documentIds = new ArrayList<>();  
 DocumentId documentId = new DocumentId();  
 documentId.setDocumentId(UUID.*randomUUID*());  
 documentIds.add(documentId);  
 documentIdsRequest.setDocuments(documentIds);  
  
 List<UUID> documentUUIds = new ArrayList<>();  
 documentUUIds.add(UUID.*randomUUID*());  
 List<Document> documentList = new ArrayList<>();  
 Document document = new Document();  
 document.setDocumentType("Some DocumentType");  
 document.setDocumentNumber(1L);  
 document.setCustomerAccountUuid(UUID.*randomUUID*());  
 documentList.add(document);  
  
 List<DocumentResponse> documentResponses = new ArrayList<>();  
 DocumentResponse documentResponse = new DocumentResponse();  
 documentResponse.setDocumentId(UUID.*randomUUID*());  
 documentResponse.setDocumentType("sample documentType");  
 documentResponse.setDocumentNumber(1L);  
 documentResponse.setCustomerAccountUuid(UUID.*randomUUID*());  
 documentResponses.add(documentResponse);  
  
 //Stub the mock  
 *when*(documentGeneratorEventStoreServiceImpl.getDocumentUUIDsInList(documentIdsRequest.getDocuments())).thenReturn(documentUUIds);  
 *when*(documentRepository.findByDocumentIdIn(documentUUIds)).thenReturn(documentList);  
 *when*(eventMapper.mapDocumentEntityToDocumentData(documentList)).thenReturn(documentResponses);  
  
  
 //Act  
 List<Document> result = documentRepository.findByDocumentIdIn(documentUUIds);  
 List<DocumentResponse> mapperResult = eventMapper.mapDocumentEntityToDocumentData(documentList);  
  
  
 //Validate  
 *assertEquals*(documentList, result);  
 *assertNotNull*(result);  
 *assertEquals*(documentResponses, mapperResult);  
 *verify*(documentRepository.findByDocumentIdIn(documentUUIds), *times*(1));  
 *verify*(eventMapper.mapDocumentEntityToDocumentData(documentList), *times*(1));  
 }  
 //Scenario 2- for no Response  
 @Test  
 public void testFetchDocumentsByDocumentIdsNull() {  
 //Arrange  
 DocumentIdsRequest documentIdsRequest = *mock*(DocumentIdsRequest.class);  
 List<DocumentId> documentIds = new ArrayList<>();  
 DocumentId documentId = new DocumentId();  
 documentId.setDocumentId(UUID.*randomUUID*());  
 documentIds.add(documentId);  
 documentIdsRequest.setDocuments(documentIds);  
  
 List<UUID> documentUUIds = new ArrayList<>();  
 documentUUIds.add(null);  
 List<Document> documentList = new ArrayList<>();  
 Document document = new Document();  
 document.setDocumentType("Some DocumentType");  
 document.setDocumentNumber(1L);  
 document.setCustomerAccountUuid(UUID.*randomUUID*());  
 documentList.add(document);  
  
 List<DocumentResponse> documentResponses = new ArrayList<>();  
 DocumentResponse documentResponse = new DocumentResponse();  
 documentResponse.setDocumentId(UUID.*randomUUID*());  
 documentResponse.setDocumentType("sample documentType");  
 documentResponse.setDocumentNumber(1L);  
 documentResponse.setCustomerAccountUuid(UUID.*randomUUID*());  
 documentResponses.add(documentResponse);  
  
 //Stub the mock  
 *when*(documentGeneratorEventStoreServiceImpl.getDocumentUUIDsInList(documentIdsRequest.getDocuments())).thenReturn(documentUUIds);  
 *when*(documentRepository.findByDocumentIdIn(documentUUIds)).thenReturn(Collections.*emptyList*());  
 *when*(eventMapper.mapDocumentEntityToDocumentData(documentList)).thenReturn(documentResponses);  
  
  
 //Act  
 List<Document> result = documentRepository.findByDocumentIdIn(documentUUIds);  
  
  
 //Validate  
 *assertNull*(result);  
 *assertTrue*(result.isEmpty());  
 *verify*(documentRepository.findByDocumentIdIn(documentUUIds), *times*(1));  
  
 }  
  
 // Scenario 3-for Null Document  
 @Test  
 public void testFetchDocumentsByDocumentIds\_NullDocument() {  
 //Arrange  
 DocumentIdsRequest documentIdsRequest = *mock*(DocumentIdsRequest.class);  
 documentIdsRequest.setDocuments(null);  
  
 List<UUID> documentUUIds = new ArrayList<>();  
 documentUUIds.add(UUID.*randomUUID*());  
 List<Document> documentList = new ArrayList<>();  
 Document document = new Document();  
 document.setDocumentType("Some DocumentType");  
 document.setDocumentNumber(1L);  
 document.setCustomerAccountUuid(UUID.*randomUUID*());  
 documentList.add(document);  
  
 List<DocumentResponse> documentResponses = new ArrayList<>();  
 DocumentResponse documentResponse = new DocumentResponse();  
 documentResponse.setDocumentId(UUID.*randomUUID*());  
 documentResponse.setDocumentType("sample documentType");  
 documentResponse.setDocumentNumber(1L);  
 documentResponse.setCustomerAccountUuid(UUID.*randomUUID*());  
 documentResponses.add(documentResponse);  
  
 //Stub the mock  
 *when*(documentGeneratorEventStoreServiceImpl.getDocumentUUIDsInList(documentIdsRequest.getDocuments())).thenReturn(Collections.*emptyList*());  
 *when*(documentRepository.findByDocumentIdIn(documentUUIds)).thenReturn(documentList);  
 *when*(eventMapper.mapDocumentEntityToDocumentData(documentList)).thenReturn(documentResponses);  
  
  
 //Act  
 List<UUID> result = documentGeneratorEventStoreServiceImpl.getDocumentUUIDsInList(documentIdsRequest.getDocuments());  
  
  
 //Validate  
 *assertNull*(result);  
 *assertTrue*(result.isEmpty());  
 }  
 // Scenario 4 -for null Document UUID's  
 @Test  
 public void testFetchDocumentsByDocumentIds\_nullDocumentUUID() {  
 //Arrange  
 DocumentIdsRequest documentIdsRequest = *mock*(DocumentIdsRequest.class);  
  
 List<UUID> documentUUIds = null;  
  
 //Stub the mock  
 *when*(documentRepository.findByDocumentIdIn(null)).thenThrow(new BadRequestException("Document UUId is null"));  
  
 //Act  
 List<Document> result = documentRepository.findByDocumentIdIn(null);  
  
 //Validate  
 *assertThrows*(BadRequestException.class,()->{documentRepository.findByDocumentIdIn(null);}  
 );  
 }  
  
 // Scenario 5 -for null DocumentIds request  
 @Test  
 public void testFetchDocumentsByDocumentIds\_nullDocumentIdsRequest() {  
 //Arrange  
 DocumentIdsRequest documentIdsRequest = *mock*(DocumentIdsRequest.class);  
 documentIdsRequest=null;  
  
 //Stub the mock  
 *when*(documentGeneratorEventStoreServiceImpl.getDocumentUUIDsInList(documentIdsRequest.getDocuments())).thenReturn(Collections.*emptyList*());  
  
  
 //Act  
 List<UUID> result = documentGeneratorEventStoreServiceImpl.getDocumentUUIDsInList(documentIdsRequest.getDocuments());  
  
 //Validate  
 *assertTrue*(result.isEmpty());  
  
 }  
  
 //Scenario 6-for Internal error  
  
 @Test  
 public void testFetchDocumentsByDocumentIds\_InternalError() {  
 //Arrange  
 DocumentIdsRequest documentIdsRequest = *mock*(DocumentIdsRequest.class);  
 List<DocumentId> documentIds = new ArrayList<>();  
 DocumentId documentId = new DocumentId();  
 documentId.setDocumentId(UUID.*randomUUID*());  
 documentIds.add(documentId);  
 documentIdsRequest.setDocuments(documentIds);  
  
 List<UUID> documentUUIds = new ArrayList<>();  
 documentUUIds.add(UUID.*randomUUID*());  
 List<Document> documentList = new ArrayList<>();  
 Document document = new Document();  
 document.setDocumentType("Some DocumentType");  
 document.setDocumentNumber(1L);  
 document.setCustomerAccountUuid(UUID.*randomUUID*());  
 documentList.add(document);  
  
 List<DocumentResponse> documentResponses = new ArrayList<>();  
 DocumentResponse documentResponse = new DocumentResponse();  
 documentResponse.setDocumentId(UUID.*randomUUID*());  
 documentResponse.setDocumentType("sample documentType");  
 documentResponse.setDocumentNumber(1L);  
 documentResponse.setCustomerAccountUuid(UUID.*randomUUID*());  
 documentResponses.add(documentResponse);  
  
 //Stub the mock  
 *when*(documentGeneratorEventStoreServiceImpl.getDocumentUUIDsInList(documentIdsRequest.getDocuments())).thenReturn(documentUUIds);  
 *when*(documentRepository.findByDocumentIdIn(documentUUIds).isEmpty()).thenThrow(new RuntimeException("Server Can't be reached"));  
  
  
 //Act  
 List<Document> result = documentRepository.findByDocumentIdIn(documentUUIds);  
  
  
 //Validate  
 *assertThrows*(RuntimeException.class,()->{documentRepository.findByDocumentIdIn(documentUUIds).isEmpty();  
 });  
  
  
 }  
 //Scenario 7-for getDocumentUUIDsInList success scenario  
  
 @Test  
 public void testGetDocumentUUIDsInList(){  
 List<DocumentId> documentIds=new ArrayList<>();  
 DocumentId documentId=new DocumentId();  
 documentId.setDocumentId(UUID.*randomUUID*());  
 documentIds.add(documentId);  
  
 //Act  
 List<UUID> documentUUIDs=documentGeneratorEventStoreServiceImpl.getDocumentUUIDsInList(documentIds);  
  
  
 *assertEquals*(1,documentUUIDs.size());  
 }  
  
 //Scenario 8-emptyList  
 @Test  
 public void testGetDocumentUUIDsInListEmptyList(){  
 List<DocumentId> documentIds=new ArrayList<>();  
  
  
 //Act  
 List<UUID> documentUUIDs=documentGeneratorEventStoreServiceImpl.getDocumentUUIDsInList(documentIds);  
  
  
 *assertTrue*(documentUUIDs.isEmpty());  
 }  
  
 //Scenario 9-null input  
 @Test  
 public void testGetDocumentUUIDsInListNullInput(){  
 List<DocumentId> documentIds=null;  
  
  
 //Act  
 List<UUID> documentUUIDs=documentGeneratorEventStoreServiceImpl.getDocumentUUIDsInList(documentIds);  
  
  
 *assertTrue*(documentUUIDs.isEmpty());  
 }  
 //scenario 10- for multiple documentIds  
  
 @Test  
 public void testGetDocumentUUIDsInList\_MultipleDocumentIds(){  
 List<DocumentId> documentIds=new ArrayList<>();  
 DocumentId documentId=new DocumentId();  
 documentId.setDocumentId(UUID.*randomUUID*());  
 documentIds.add(documentId);  
 documentId.setDocumentId(UUID.*randomUUID*());  
 documentIds.add(documentId);  
  
 //Act  
 List<UUID> documentUUIDs=documentGeneratorEventStoreServiceImpl.getDocumentUUIDsInList(documentIds);  
  
  
 *assertEquals*(2,documentUUIDs.size());  
 }  
 //----------------------------------------------------------------------------------------------  
 //method 11  
 //scenario 1 -Success scenario  
 @Test  
 public void testSaveDocumentDetails() {  
 // Arrange  
 DocumentRequest documentRequest = *mock*(DocumentRequest.class);  
 documentRequest.setDocumentNumber(123L);  
 documentRequest.setDocumentId(UUID.*randomUUID*());  
  
  
 Long eventId = 1L;  
 EventResponse eventResponse = new EventResponse();  
 eventResponse.setEventDataResponse(null);  
 Document document = new Document();  
 document.setDocumentId(UUID.*randomUUID*());  
 document.setDocumentNumber(123L);  
 DmEventDocument dmEventDocument = new DmEventDocument();  
 dmEventDocument.setEventId(eventId);  
 dmEventDocument.setDocumentToEventDocument(document);  
 dmEventDocument.setEventDocumentId(10L);  
  
 Long documentNumber = document.getDocumentNumber();  
 EventDataResponse eventDataResponse = new EventDataResponse();  
 eventDataResponse.setEventId(String.*valueOf*(1L));  
 eventDataResponse.setDocumentRequest(documentRequest);  
  
 //stub the mock  
 *when*(eventMapper.eventRequestToDocumentEntity(documentRequest)).thenReturn(document);  
 *when*(documentGeneratorEventStoreServiceImpl.getDocumentNumber()).thenReturn(documentNumber);  
 *when*(documentRepository.save(document)).thenReturn(document);  
 *when*(dmEventDocumentRepository.save(dmEventDocument)).thenReturn(dmEventDocument);  
  
  
 //Act  
 Document result = documentRepository.save(document);  
 DmEventDocument dmResult = dmEventDocumentRepository.save(dmEventDocument);  
 Long documentNumberResult = documentGeneratorEventStoreServiceImpl.getDocumentNumber();  
  
  
 //validate  
 *assertEquals*(result, document);  
 *assertEquals*(dmResult, dmEventDocument);  
 *assertEquals*(result.getDocumentId(), document.getDocumentId());  
 *assertEquals*(documentNumberResult, documentNumber);  
 *assertEquals*(dmResult.getEventId(), eventId);  
 *assertEquals*(dmResult.getDocumentToEventDocument(), document);  
 }  
  
 //scenario 2 --for Null DocumentRequest  
 @Test  
 public void testSaveDocumentDetails\_NullDocumentRequest() {  
 DocumentRequest documentRequest = *mock*(DocumentRequest.class);  
 documentRequest = null;  
  
 *when*(eventMapper.eventRequestToDocumentEntity(null)).thenThrow(new IllegalArgumentException("invalid documentRequest"));  
  
 //Act  
 Document result = eventMapper.eventRequestToDocumentEntity(null);  
  
 //validate  
 *assertThrows*(IllegalArgumentException.class, () -> {  
 eventMapper.eventRequestToDocumentEntity(null);  
 });  
  
 }  
  
 //Scenario 3-for for null Document  
 @Test  
 public void testSaveDocumentDetails\_NullDocument() {  
 // Arrange  
 DocumentRequest documentRequest = *mock*(DocumentRequest.class);  
 documentRequest.setDocumentNumber(123L);  
 documentRequest.setDocumentId(UUID.*randomUUID*());  
  
 Long eventId = 1L;  
 EventResponse eventResponse = new EventResponse();  
 eventResponse.setEventDataResponse(null);  
 DmEventDocument dmEventDocument = new DmEventDocument();  
 dmEventDocument.setEventId(eventId);  
 dmEventDocument.setEventDocumentId(10L);  
 EventDataResponse eventDataResponse = new EventDataResponse();  
 eventDataResponse.setEventId(String.*valueOf*(1L));  
 eventDataResponse.setDocumentRequest(documentRequest);  
 Document document = null;  
  
 //stub the mock  
 *when*(documentRepository.save(document)).thenThrow(new IllegalArgumentException("Document is null"));  
 *when*(dmEventDocumentRepository.save(dmEventDocument)).thenReturn(dmEventDocument);  
  
  
 //Act  
 Document result = documentRepository.save(document);  
 DmEventDocument dmResult = dmEventDocumentRepository.save(dmEventDocument);  
 Long documentNumberResult = documentGeneratorEventStoreServiceImpl.getDocumentNumber();  
  
  
 //validate  
 *assertEquals*(dmResult, dmEventDocument);  
 *assertThrows*(IllegalArgumentException.class, () -> {  
 documentRepository.save(document);  
 });  
 }  
  
 //Scenario 4 -Setting the EventId as Null  
 @Test  
 public void testSaveDocumentDetails\_EventIdNull() {  
 // Arrange  
 DocumentRequest documentRequest = *mock*(DocumentRequest.class);  
 documentRequest.setDocumentNumber(123L);  
 documentRequest.setDocumentId(UUID.*randomUUID*());  
  
 Long eventId = null;  
  
 EventResponse eventResponse = new EventResponse();  
 eventResponse.setEventDataResponse(null);  
 Document document = new Document();  
 document.setDocumentId(UUID.*randomUUID*());  
 document.setDocumentNumber(123L);  
 DmEventDocument dmEventDocument = new DmEventDocument();  
 dmEventDocument.setEventId(null);  
 dmEventDocument.setEventDocumentId(10L);  
  
 Long documentNumber = document.getDocumentNumber();  
 EventDataResponse eventDataResponse = new EventDataResponse();  
 eventDataResponse.setDocumentRequest(documentRequest);  
  
 //stub the mock  
 *when*(eventMapper.eventRequestToDocumentEntity(documentRequest)).thenReturn(document);  
 *when*(documentGeneratorEventStoreServiceImpl.getDocumentNumber()).thenReturn(documentNumber);  
 *when*(documentRepository.save(document)).thenReturn(document);  
 *when*(dmEventDocumentRepository.save(dmEventDocument)).thenReturn(dmEventDocument);  
  
  
 //Act  
 Document result = documentRepository.save(document);  
 DmEventDocument dmResult = dmEventDocumentRepository.save(dmEventDocument);  
 Long documentNumberResult = documentGeneratorEventStoreServiceImpl.getDocumentNumber();  
  
  
 //validate  
 *assertNull*(dmResult.getEventDocumentId());  
 *assertEquals*(result, document);  
 *assertEquals*(dmResult, dmEventDocument);  
 }  
  
 //Scenario 5 -Setting document to event document as Null  
 @Test  
 public void testSaveDocumentDetails\_EventDocumentNull() {  
 // Arrange  
 DocumentRequest documentRequest = *mock*(DocumentRequest.class);  
 documentRequest.setDocumentNumber(123L);  
 documentRequest.setDocumentId(UUID.*randomUUID*());  
  
  
 Long eventId = 1L;  
 EventResponse eventResponse = new EventResponse();  
 eventResponse.setEventDataResponse(null);  
 Document document = new Document();  
 document.setDocumentId(UUID.*randomUUID*());  
 document.setDocumentNumber(123L);  
 DmEventDocument dmEventDocument = new DmEventDocument();  
 dmEventDocument.setEventId(eventId);  
 dmEventDocument.setEventDocumentId(10L);  
 dmEventDocument.setDocumentToEventDocument(null);  
 Long documentNumber = document.getDocumentNumber();  
 EventDataResponse eventDataResponse = new EventDataResponse();  
 eventDataResponse.setEventId(String.*valueOf*(1L));  
 eventDataResponse.setDocumentRequest(documentRequest);  
  
 //stub the mock  
 *when*(eventMapper.eventRequestToDocumentEntity(documentRequest)).thenReturn(document);  
 *when*(documentGeneratorEventStoreServiceImpl.getDocumentNumber()).thenReturn(documentNumber);  
 *when*(documentRepository.save(document)).thenReturn(document);  
 *when*(dmEventDocumentRepository.save(dmEventDocument)).thenReturn(dmEventDocument);  
  
  
 //Act  
 Document result = documentRepository.save(document);  
 DmEventDocument dmResult = dmEventDocumentRepository.save(dmEventDocument);  
  
  
 //validate  
 *assertNull*(dmResult.getDocumentToEventDocument());  
 *assertEquals*(result, document);  
 *assertEquals*(dmResult, dmEventDocument);  
  
 }  
  
 //Scenario 6 -Internal Server error for documentRepository  
 @Test  
 public void testSaveDocumentDetails\_Internal\_server\_Error() {  
 // Arrange  
 DocumentRequest documentRequest = *mock*(DocumentRequest.class);  
 documentRequest.setDocumentNumber(123L);  
 documentRequest.setDocumentId(UUID.*randomUUID*());  
  
  
 Long eventId = 1L;  
 EventResponse eventResponse = new EventResponse();  
 eventResponse.setEventDataResponse(null);  
 Document document = new Document();  
 document.setDocumentId(UUID.*randomUUID*());  
 document.setDocumentNumber(123L);  
 DmEventDocument dmEventDocument = new DmEventDocument();  
 dmEventDocument.setEventId(eventId);  
 dmEventDocument.setEventDocumentId(10L);  
  
 Long documentNumber = document.getDocumentNumber();  
 EventDataResponse eventDataResponse = new EventDataResponse();  
 eventDataResponse.setEventId(String.*valueOf*(1L));  
 eventDataResponse.setDocumentRequest(documentRequest);  
  
 //stub the mock  
 *when*(eventMapper.eventRequestToDocumentEntity(documentRequest)).thenReturn(document);  
 *when*(documentGeneratorEventStoreServiceImpl.getDocumentNumber()).thenReturn(documentNumber);  
 *when*((documentRepository.save(document)).getDocumentExtData().isNull()).thenThrow(new RuntimeException("server can't ne reached"));  
 *when*(dmEventDocumentRepository.save(dmEventDocument)).thenReturn(dmEventDocument);  
  
  
 //Act  
 Document result = documentRepository.save(document);  
 DmEventDocument dmResult = dmEventDocumentRepository.save(dmEventDocument);  
  
 //validate  
 *assertThrows*(RuntimeException.class, () -> {  
 (documentRepository.save(document)).getDocumentExtData().isNull();  
 });  
 *assertEquals*(dmResult, dmEventDocument);  
 *assertEquals*(result.getDocumentId(), document.getDocumentId());  
  
  
 }  
  
 //Scenario 7 -Internal Server error for dmEventDocumentRepository  
 @Test  
 public void testSaveDocumentDetails\_server\_Error() {  
 // Arrange  
 DocumentRequest documentRequest = *mock*(DocumentRequest.class);  
 documentRequest.setDocumentNumber(123L);  
 documentRequest.setDocumentId(UUID.*randomUUID*());  
  
  
 Long eventId = 1L;  
 EventResponse eventResponse = new EventResponse();  
 eventResponse.setEventDataResponse(null);  
 Document document = new Document();  
 document.setDocumentId(UUID.*randomUUID*());  
 document.setDocumentNumber(123L);  
 DmEventDocument dmEventDocument = new DmEventDocument();  
 dmEventDocument.setEventId(eventId);  
 dmEventDocument.setEventDocumentId(10L);  
  
 Long documentNumber = document.getDocumentNumber();  
 EventDataResponse eventDataResponse = new EventDataResponse();  
 eventDataResponse.setEventId(String.*valueOf*(1L));  
 eventDataResponse.setDocumentRequest(documentRequest);  
  
 //stub the mock  
 *when*(eventMapper.eventRequestToDocumentEntity(documentRequest)).thenReturn(document);  
 *when*(documentGeneratorEventStoreServiceImpl.getDocumentNumber()).thenReturn(documentNumber);  
 *when*(documentRepository.save(document)).thenReturn(document);  
 *when*(dmEventDocumentRepository.save(dmEventDocument)).thenThrow(new RuntimeException("server can't be reached"));  
  
  
 //Act  
 Document result = documentRepository.save(document);  
 DmEventDocument dmResult = dmEventDocumentRepository.save(dmEventDocument);  
  
 //validate  
 *assertThrows*(RuntimeException.class, () -> {  
 dmEventDocumentRepository.save(dmEventDocument);  
 });  
 *assertEquals*(dmResult, dmEventDocument);  
 *assertEquals*(result.getDocumentId(), document.getDocumentId());  
  
  
 }  
 //Scenario 8- if dmEventDocument is null  
 @Test  
 public void testSaveDocumentDetails\_NullDmEventDocument() {  
 // Arrange  
 DocumentRequest documentRequest = *mock*(DocumentRequest.class);  
 documentRequest.setDocumentNumber(123L);  
 documentRequest.setDocumentId(UUID.*randomUUID*());  
  
  
 Long eventId = 1L;  
 EventResponse eventResponse = new EventResponse();  
 eventResponse.setEventDataResponse(null);  
 Document document = new Document();  
 document.setDocumentId(UUID.*randomUUID*());  
 document.setDocumentNumber(123L);  
 DmEventDocument dmEventDocument = null;  
  
  
 Long documentNumber = document.getDocumentNumber();  
 EventDataResponse eventDataResponse = new EventDataResponse();  
 eventDataResponse.setEventId(String.*valueOf*(1L));  
 eventDataResponse.setDocumentRequest(documentRequest);  
  
 //stub the mock  
 *when*(eventMapper.eventRequestToDocumentEntity(documentRequest)).thenReturn(document);  
 *when*(documentGeneratorEventStoreServiceImpl.getDocumentNumber()).thenReturn(documentNumber);  
 *when*(documentRepository.save(document)).thenReturn(document);  
 *when*(dmEventDocumentRepository.save(null)).thenReturn(new IllegalArgumentException("dmEventDocument is null"));  
  
  
 //Act  
 Document result = documentRepository.save(document);  
 DmEventDocument dmResult = dmEventDocumentRepository.save(null);  
 Long documentNumberResult = documentGeneratorEventStoreServiceImpl.getDocumentNumber();  
  
  
 //validate  
 *assertEquals*(result, document);  
 *assertNull*(dmResult);  
 *assertEquals*(result.getDocumentId(), document.getDocumentId());  
  
 }  
//----------------------------------------------------------------------------------------------------

//Service Class ResponseMapper

package com.project.bos.dg.datastore.service.impl;  
  
import com.project.bos.dg.datastore.constants.DocumentGeneratorEventStoreConstants;  
import com.project.bos.dg.datastore.mapper.EventMapper;  
import com.project.bos.dg.datastore.model.request.CreateDocumentServiceRequestStatus;  
import com.project.bos.dg.datastore.model.request.DocumentServiceRequestData;  
import com.project.bos.dg.datastore.model.response.DocumentRetrievalResponse;  
import com.project.bos.dg.datastore.model.response.common.DocumentResponse;  
import com.project.bos.dg.datastore.model.response.documents.DocumentIdsResponse;  
import org.junit.jupiter.api.Test;  
import org.junit.runner.RunWith;  
import org.mockito.InjectMocks;  
import org.mockito.Mock;  
import org.mockito.Mockito;  
import org.mockito.junit.MockitoJUnitRunner;  
import org.springframework.http.HttpStatus;  
import org.springframework.http.MediaType;  
import org.springframework.http.ResponseEntity;  
import org.springframework.web.client.HttpClientErrorException;  
  
import java.util.ArrayList;  
import java.util.List;  
  
import static javax.management.Query.*eq*;  
import static org.junit.jupiter.api.Assertions.\*;  
import static org.mockito.Mockito.\*;  
  
@RunWith(MockitoJUnitRunner.class)  
class ResponseMapperTest {  
 @InjectMocks  
 private ResponseMapper responseMapper;  
  
 @Mock  
 private EventMapper eventMapper;  
  
 //method 3  
 //Scenario 1-success Scenario  
 @Test  
 public void testSetDocumentResponse() {  
 CreateDocumentServiceRequestStatus createDocumentServiceRequestStatus = new CreateDocumentServiceRequestStatus();  
 DocumentServiceRequestData documentServiceRequestDTO = new DocumentServiceRequestData();  
 DocumentRetrievalResponse documentResponse = new DocumentRetrievalResponse();  
 documentResponse.setCode(HttpStatus.*OK*.value());  
 documentResponse.setStatus(String.*valueOf*(HttpStatus.*OK*.value()));  
 documentResponse.setMessage(DocumentGeneratorEventStoreConstants.*NOT\_SAVED*);  
  
  
 // Call the method to be tested  
 eventMapper.updateCreateDocumentServiceRequestStatusToDocumentResponse(  
 createDocumentServiceRequestStatus, documentResponse);  
  
 // Act  
 responseMapper.setDocumentResponse(createDocumentServiceRequestStatus, documentServiceRequestDTO, documentResponse);  
  
  
 *assertNotNull*(documentResponse.getCode());  
 *assertNotNull*(documentResponse.getStatus());  
 *assertNotNull*(documentResponse.getMessage());  
 *assertNotNull*(documentResponse.getDocumentDataResponse());  
 *assertEquals*(documentServiceRequestDTO, documentResponse.getDocumentDataResponse().getDocumentServiceRequestDTO());  
  
 // You can use verify() to ensure that the method was called with the expected parameters  
 *verify*(eventMapper).updateCreateDocumentServiceRequestStatusToDocumentResponse(  
 createDocumentServiceRequestStatus, documentResponse);  
 }  
  
 //Scenario 2-for Null DTO  
 @Test  
 public void testSetDocumentResponse\_nullDTO() {  
 CreateDocumentServiceRequestStatus createDocumentServiceRequestStatus = new CreateDocumentServiceRequestStatus();  
 DocumentServiceRequestData documentServiceRequestDTO = null;  
 DocumentRetrievalResponse documentResponse = new DocumentRetrievalResponse();  
  
  
 // Call the method to be tested  
 eventMapper.updateCreateDocumentServiceRequestStatusToDocumentResponse(  
 createDocumentServiceRequestStatus, documentResponse);  
  
 // Act  
 responseMapper.setDocumentResponse(createDocumentServiceRequestStatus, null, documentResponse);  
  
 *assertEquals*(HttpStatus.*OK*.value(), documentResponse.getCode());  
 *assertEquals*(String.*valueOf*(HttpStatus.*OK*.value()), documentResponse.getStatus());  
 *assertEquals*(DocumentGeneratorEventStoreConstants.*NOT\_SAVED*, documentResponse.getMessage());  
 *assertNull*(documentResponse.getDocumentDataResponse());  
  
  
 // You can use verify() to ensure that the method was called with the expected parameters  
 *verify*(eventMapper).updateCreateDocumentServiceRequestStatusToDocumentResponse(  
 createDocumentServiceRequestStatus, documentResponse);  
 }  
  
 //Scenario 3 - NullDto,Null documentResponse  
 @Test  
 public void testSetDocumentResponseWithNullDTOAndNullResponse() {  
 // Arrange  
 CreateDocumentServiceRequestStatus createDocumentServiceRequestStatus = new CreateDocumentServiceRequestStatus();  
 DocumentServiceRequestData documentServiceRequestDTO = null;  
 DocumentRetrievalResponse documentResponse = null;  
  
 //act  
 responseMapper.setDocumentResponse(createDocumentServiceRequestStatus, null, null)  
  
  
 // Ensure that the method doesn't throw any exceptions in this scenario  
 *assertDoesNotThrow*(() -> responseMapper.setDocumentResponse(createDocumentServiceRequestStatus, null, null));  
 }  
//----------------------------------------------------------------------------------------------------  
  
 //Method 5  
 //Scenario 1-success Scenario  
 @Test  
 public void testCreateResponseEntity(){  
 // Create a list of DocumentResponse  
 List<DocumentResponse> documentDataResponse = new ArrayList<>();  
 DocumentResponse response= new DocumentResponse();  
 documentDataResponse.add(response);  
  
 String correlationId = "testCorrelationId";  
 String applicationLabel = "testApplicationLabel";  
  
 // Call the method  
 ResponseEntity<DocumentIdsResponse> responseEntity = responseMapper.createResponseEntity(  
 documentDataResponse,  
 correlationId,  
 applicationLabel  
 );  
  
 DocumentIdsResponse responseBody = responseEntity.getBody();  
 // validate  
 *assertEquals*(HttpStatus.*OK*, responseEntity.getStatusCode());  
 *assertEquals*(MediaType.*APPLICATION\_JSON*, responseEntity.getHeaders().getContentType());  
  
  
 *assertEquals*(HttpStatus.*OK*.value(), responseBody.getCode());  
 *assertEquals*(correlationId, responseBody.getCorrelationId());  
 *assertEquals*(applicationLabel, responseBody.getApplicationLabel());  
 *assertEquals*(String.*valueOf*(HttpStatus.*OK*.value()), responseBody.getStatus());  
 *assertEquals*(DocumentGeneratorEventStoreConstants.*SUCCESS*, responseBody.getMessage());  
 *assertEquals*(documentDataResponse, responseBody.getDocumentData());  
 }  
  
  
 //Scenario 2-for differentHttpStatusCode  
 @Test  
 public void testCreateResponseEntity\_differentHttpStatusCode() {  
 // Create a list of DocumentResponse  
 List<DocumentResponse> documentDataResponse = new ArrayList<>();  
 DocumentResponse response = new DocumentResponse();  
 documentDataResponse.add(response);  
  
 String correlationId = "testCorrelationId";  
 String applicationLabel = "testApplicationLabel";  
  
 // Call the method  
 ResponseEntity<DocumentIdsResponse> responseEntity = responseMapper.createResponseEntity(  
 documentDataResponse,  
 correlationId,  
 applicationLabel  
 );  
 DocumentIdsResponse responseBody = responseEntity.getBody();  
 responseBody.setCode(HttpStatus.*BAD\_REQUEST*.value());  
 // validate  
 *assertEquals*(HttpStatus.*BAD\_REQUEST*, responseEntity.getStatusCode());  
 *assertEquals*(MediaType.*APPLICATION\_JSON*, responseEntity.getHeaders().getContentType());  
 }  
 //Scenario 3-for differentContentType  
 @Test  
 public void testCreateResponseEntity\_differentContentType() {  
 // Create a list of DocumentResponse  
 List<DocumentResponse> documentDataResponse = new ArrayList<>();  
 DocumentResponse response = new DocumentResponse();  
 documentDataResponse.add(response);  
  
 String correlationId = "testCorrelationId";  
 String applicationLabel = "testApplicationLabel";  
  
 // Call the method  
 ResponseEntity<DocumentIdsResponse> responseEntity = responseMapper.createResponseEntity(  
 documentDataResponse,  
 correlationId,  
 applicationLabel  
 );  
 responseEntity.getHeaders().setContentType(MediaType.*APPLICATION\_XML*);  
  
 // validate  
 *assertEquals*(MediaType.*APPLICATION\_XML*, responseEntity.getHeaders().getContentType());  
 }  
 //---------------------------------------------------------------------------------------------------  
}