package com.project.bos.dg.datastore.service.impl;  
  
import com.fasterxml.jackson.core.JsonProcessingException;  
import com.project.bos.dg.datastore.mapper.EventMapper;  
import com.project.bos.dg.datastore.model.entity.DmEvent;  
import com.project.bos.dg.datastore.model.entity.DmEventNotify;  
import com.project.bos.dg.datastore.model.entity.DmEventStatus;  
import com.project.bos.dg.datastore.model.request.CreateEventStatusRequest;  
import com.project.bos.dg.datastore.model.request.EventRequest;  
import com.project.bos.dg.datastore.model.request.EventStatusRequest;  
import com.project.bos.dg.datastore.model.response.EventResponse;  
  
import com.project.bos.dg.datastore.model.request.CreateEventStatusRequest;  
import com.project.bos.dg.datastore.model.request.EventStatusRequest;  
import com.project.bos.dg.datastore.model.request.EventErrorRequest;  
import com.project.bos.dg.datastore.model.request.EventNotifyDataRequest;  
  
  
import com.project.bos.dg.datastore.repository.DmEventNotifyRepository;  
import com.project.bos.dg.datastore.repository.DmEventRepository;  
import com.project.bos.dg.datastore.repository.DmEventStatusRepository;  
import com.project.bos.dg.datastore.service.DocumentGeneratorEventStoreService;  
import org.junit.jupiter.api.BeforeEach;  
import org.junit.jupiter.api.RepeatedTest;  
import org.junit.jupiter.api.Test;  
import org.junit.runner.RunWith;  
import org.mockito.InjectMocks;  
import org.mockito.Mock;  
import org.mockito.MockitoAnnotations;  
import org.mockito.stubbing.OngoingStubbing;  
import org.springframework.boot.test.context.SpringBootTest;  
import org.springframework.dao.DataAccessException;  
import org.springframework.dao.DataIntegrityViolationException;  
import org.springframework.http.HttpStatus;  
import org.springframework.test.context.junit4.SpringRunner;  
  
import java.time.Duration;  
import java.util.Optional;  
import java.util.concurrent.CountDownLatch;  
import java.util.concurrent.ExecutorService;  
import java.util.concurrent.Executors;  
import java.util.concurrent.TimeUnit;  
import java.util.logging.Logger;  
  
import static org.junit.jupiter.api.Assertions.\*;  
import static org.mockito.ArgumentMatchers.any;  
import static org.mockito.Mockito.\*;  
@RunWith(SpringRunner.class)  
@SpringBootTest  
class DocumentGeneratorEventStoreServiceImplTest {  
  
  
 @InjectMocks  
 private DocumentGeneratorEventStoreService eventStoreService;  
  
 @Mock  
 private EventStoreHelper eventStoreHelper;  
  
 @Mock  
 private EventMapper eventMapper;  
  
 @Mock  
 private ResponseMapper responseMapper;  
  
 @Mock  
 private DmEventRepository dmEventRepository;  
  
 @Mock  
 private DmEventStatusRepository dmEventStatusRepository;  
  
 @Mock  
 private Logger logger;  
  
 @Mock  
 private DmEventNotifyRepository dmEventNotifyRepository;  
  
 //Method - 1 Event creation success testcase-1  
 @Test  
 public void testSaveEvent\_EventCreation\_Success() throws JsonProcessingException {  
 // Create a sample EventRequest  
 EventRequest eventRequest = new EventRequest();  
 eventRequest.setCorrelationId("12345");  
  
 // Create a sample EventResponse  
 EventResponse eventResponse = new EventResponse();  
 eventResponse.setCode(Integer.valueOf("SUCCESS"));  
  
 // Mock the behavior of eventStoreHelper to return the eventResponse  
 when(eventStoreHelper.getEventDataResponse(eventRequest, eventResponse))  
 .thenReturn(Optional.of(eventResponse));  
  
 // Call the saveEvent method  
 Optional<EventResponse> result = DocumentGeneratorEventStoreService.saveEvent(eventRequest);  
  
 // Verify that the service correctly saves the event and returns the expected response  
 assertTrue(result.isPresent());  
 assertEquals("SUCCESS", result.get().getCode());  
  
 // Verify that eventStoreHelper.getEventDataResponse was called with the expected arguments  
 verify(eventStoreHelper, times(1)).getEventDataResponse(eventRequest, eventResponse);  
 }  
  
 //Method -2 testcase-2 testSaveEvent\_AlreadyExists  
 @Test  
 public void testSaveEvent\_AlreadyExists() throws JsonProcessingException {  
 // Arrange  
 EventRequest eventRequest = new EventRequest();  
 eventRequest.setCorrelationId("correlationId");  
  
 EventResponse expectedResponse = new EventResponse();  
 expectedResponse.setCode(HttpStatus.OK.value());  
  
 when(eventStoreHelper.getEventDataResponse(any(EventRequest.class), any(EventResponse.class)))  
 .thenReturn(Optional.of(expectedResponse));  
  
 // Act  
 Optional<EventResponse> result = DocumentGeneratorEventStoreService.saveEvent(eventRequest);  
  
 // Assert  
 assertTrue(result.isPresent());  
 assertEquals(HttpStatus.OK.value(), result.get().getCode());  
 }  
  
 //Method -1 testcase -3testSaveEvent\_EventUpdate\_Success  
 @Test  
 public void testSaveEvent\_EventUpdate\_Success() throws JsonProcessingException {  
 // Create a sample EventRequest  
 EventRequest eventRequest = new EventRequest();  
 eventRequest.setCorrelationId("12345");  
  
 // Create a sample EventResponse  
 EventResponse eventResponse = new EventResponse();  
 eventResponse.setCode(Integer.valueOf("SUCCESS"));  
  
 // Mock the behavior of eventStoreHelper to return the eventResponse  
 when(eventStoreHelper.getEventDataResponse(eventRequest, eventResponse))  
 .thenReturn(Optional.of(eventResponse));  
  
 // Mock the behavior of findExistingEvent to return an existing event  
 when(dmEventRepository.findById(Long.valueOf(eventRequest.getCorrelationId())))  
 .thenReturn(Optional.of(new DmEvent()));  
  
 // Call the saveEvent method  
 Optional<EventResponse> result = DocumentGeneratorEventStoreService.saveEvent(eventRequest);  
  
 // Verify that the service correctly updates the event and returns the expected response  
 assertTrue(result.isPresent());  
 assertEquals("SUCCESS", result.get().getCode());  
  
 // Verify that eventStoreHelper.getEventDataResponse was called with the expected arguments  
 verify(eventStoreHelper, times(1)).getEventDataResponse(eventRequest, eventResponse);  
  
 // Verify that dmEventRepository.findByCorrelationId was called with the expected correlationId  
 verify(dmEventRepository, times(1)).findById(Long.valueOf(eventRequest.getCorrelationId()));  
 }  
  
  
 //Method -1 Testcase-4 testDataIntegrityViolationHandling  
 @Test  
 public void testDataIntegrityViolationHandling() throws JsonProcessingException {  
 // Create a sample EventRequest  
 EventRequest eventRequest = new EventRequest();  
 eventRequest.setCorrelationId("12345");  
  
 // Mock the behavior of eventStoreHelper to throw a DataIntegrityViolationException  
 when(eventStoreHelper.getEventDataResponse(eventRequest, new EventResponse()))  
 .thenThrow(new DataIntegrityViolationException("Unique constraint violation"));  
  
 // Mock the behavior of findExistingEvent to return an existing event  
 when(dmEventRepository.findById(Long.valueOf(eventRequest.getCorrelationId())))  
 .thenReturn(Optional.of(new DmEvent()));  
  
 // Call the saveEvent method  
 Optional<EventResponse> result = DocumentGeneratorEventStoreService.saveEvent(eventRequest);  
  
 // Verify that the service correctly handles the DataIntegrityViolationException  
 assertTrue(result.isPresent());  
 assertEquals("SUCCESS", result.get().getCode()); // Ensure the event was updated  
  
 // Verify that eventStoreHelper.getEventDataResponse was called with the expected arguments  
 verify(eventStoreHelper, times(1)).getEventDataResponse(eventRequest, new EventResponse());  
  
 // Verify that dmEventRepository.findByCorrelationId was called with the expected correlationId  
 verify(dmEventRepository, times(1)).findById(Long.valueOf(eventRequest.getCorrelationId()));  
 }  
  
 //Method -1 testcase 5  
 @Test  
 public void testInvalidJsonRequestHandling() throws JsonProcessingException {  
 // Create a sample invalid JSON request that cannot be deserialized  
 EventRequest invalidEventRequest = new EventRequest();  
 String invalididField = "1234";  
 invalidEventRequest.setCorrelationId("12345");  
 invalidEventRequest.setApplicationLabel("SomeEventType"); // This field does not exist in EventRequest class  
  
 // Mock the behavior of eventStoreHelper to throw a JsonProcessingException  
 when(eventStoreHelper.getEventDataResponse(eq(invalidEventRequest), any()))  
 .thenThrow(JsonProcessingException.class);  
  
 // Call the saveEvent method with the invalid request  
 assertThrows(JsonProcessingException.class, () -> DocumentGeneratorEventStoreService.saveEvent(invalidEventRequest));  
  
 // Verify that eventStoreHelper.getEventDataResponse was called with the invalid request  
 verify(eventStoreHelper, times(1)).getEventDataResponse(eq(invalidEventRequest), any());  
 }  
  
  
 //Method -1 testcase -5 testTransactionRollback  
 @Test  
 public void testTransactionRollback() throws JsonProcessingException {  
 // Create a sample EventRequest  
 EventRequest eventRequest = new EventRequest();  
 eventRequest.setCorrelationId("12345");  
  
 // Mock the behavior of eventStoreHelper to throw an exception other than DataIntegrityViolationException  
 when(eventStoreHelper.getEventDataResponse(eventRequest, new EventResponse()))  
 .thenThrow(RuntimeException.class);  
  
 // Call the saveEvent method  
 assertThrows(RuntimeException.class, () -> DocumentGeneratorEventStoreService.saveEvent(eventRequest));  
  
 // Verify that eventStoreHelper.getEventDataResponse was called with the expected arguments  
 verify(eventStoreHelper, times(1)).getEventDataResponse(eventRequest, new EventResponse());  
  
 // Verify that dmEventRepository.save (or any database save operation) was NOT called  
 verify(dmEventRepository, never()).save(any());  
 }  
  
 //Method -1 testcases 6 testDependencyInteraction  
 @Test  
 public void testDependencyInteraction() throws JsonProcessingException {  
 // Create a sample EventRequest  
 EventRequest eventRequest = new EventRequest();  
 eventRequest.setCorrelationId("12345");  
  
 // Create a sample EventResponse  
 EventResponse eventResponse = new EventResponse();  
 eventResponse.setCode(Integer.valueOf("SUCCESS"));  
  
 // Mock the behavior of eventStoreHelper to return the eventResponse  
 when(eventStoreHelper.getEventDataResponse(eventRequest, eventResponse))  
 .thenReturn(Optional.of(eventResponse));  
  
 // Mock the behavior of dmEventRepository to return an existing event  
 when(dmEventRepository.findById(Long.valueOf(eventRequest.getCorrelationId())))  
 .thenReturn(Optional.of(new DmEvent()));  
  
 // Mock the behavior of eventMapper and responseMapper  
  
 // Call the saveEvent method  
 Optional<EventResponse> result = DocumentGeneratorEventStoreService.saveEvent(eventRequest);  
  
 // Verify that the service correctly interacts with its dependencies  
 assertTrue(result.isPresent());  
 assertEquals("SUCCESS", result.get().getCode());  
  
 // Verify interactions with dependencies  
 verify(eventStoreHelper, times(1)).getEventDataResponse(eventRequest, eventResponse);  
 verify(dmEventRepository, times(1)).findById(Long.valueOf(eventRequest.getCorrelationId()));  
 verify(eventMapper, times(1)).updateExistingEventResponse(eventRequest, eventResponse);  
 verify(responseMapper, times(1)).mapEventTOEventResponse(any(), eq(eventResponse));  
 }  
  
 //Method -1 testcase -7  
 @Test  
 public void testSaveEventWithEmptyEventRequest() throws JsonProcessingException {  
 // Arrange  
 when(eventStoreHelper.getEventDataResponse(any(EventRequest.class), any(EventResponse.class)))  
 .thenReturn(Optional.of(new EventResponse())); // Simulate a successful response  
  
 // Act  
 Optional<EventResponse> result = DocumentGeneratorEventStoreService.saveEvent(new EventRequest()); // Pass an empty EventRequest  
  
 // Assert  
 assertTrue(result.isPresent());  
 // Add more assertions to verify the content of the response  
 verify(eventStoreHelper, times(1)).getEventDataResponse(any(EventRequest.class), any(EventResponse.class));  
 }  
  
 //Method -1 testcase - 8  
 @Test  
 public void testSaveEventWithNullEventRequest() throws JsonProcessingException {  
 // Arrange  
 when(eventStoreHelper.getEventDataResponse(any(EventRequest.class), any(EventResponse.class)))  
 .thenReturn(Optional.of(new EventResponse())); // Simulate a successful response  
  
 // Act  
 Optional<EventResponse> result = DocumentGeneratorEventStoreService.saveEvent(null); // Pass a null EventRequest  
  
 // Assert  
 assertTrue(result.isPresent());  
 // Add more assertions to verify the content of the response  
 verify(eventStoreHelper, times(0)).getEventDataResponse(any(EventRequest.class), any(EventResponse.class));  
 }  
  
 //Method -1 Testcase -9  
 //to verify how the service handles various exceptions, both expected and unexpected.  
 @Test  
 public void testSaveEventWithDataIntegrityViolationException() throws JsonProcessingException, InterruptedException {  
 // Arrange  
 EventRequest eventRequest = new EventRequest();  
 String correlationId = "correlationId";  
 eventRequest.setCorrelationId(correlationId);  
 EventResponse eventResponse = new EventResponse();  
  
 when(eventStoreHelper.getEventDataResponse(eventRequest, eventResponse))  
 .thenThrow(DataIntegrityViolationException.class);  
  
 // Act and Assert  
 try {  
 DocumentGeneratorEventStoreService.saveEvent(eventRequest);  
 fail("Expected DataIntegrityViolationException was not thrown.");  
 } catch (DataIntegrityViolationException ce) {  
 // Verify that the expected exception was thrown.  
 verify(eventStoreHelper, times(1)).getEventDataResponse(eventRequest, eventResponse);  
 verify(eventMapper, never()).updateExistingEventResponse(any(), any());  
 verify(responseMapper, never()).mapEventTOEventResponse(any(), any());  
 }  
  
 }  
  
 //Method-1 Testcase:10  
 @Test  
 public void testSaveEventWithLargeCorrelationId() throws JsonProcessingException {  
 // Arrange  
 EventRequest eventRequest = new EventRequest();  
 String largeCorrelationId = generateLargeString(); // Generate a large correlation ID  
 eventRequest.setCorrelationId(largeCorrelationId);  
 EventResponse eventResponse = new EventResponse();  
  
 // Stub the behavior of eventStoreHelper  
 when(eventStoreHelper.getEventDataResponse(eventRequest, eventResponse))  
 .thenReturn(Optional.of(eventResponse));  
  
 // Act  
 Optional<EventResponse> result = DocumentGeneratorEventStoreService.saveEvent(eventRequest);  
  
 // Assert  
 verify(eventStoreHelper, times(1)).getEventDataResponse(eventRequest, eventResponse);  
 assertTrue(result.isPresent());  
 }  
//Method-1 Testcase: 11  
 @Test  
 public void testSaveEventWithSmallCorrelationId() throws JsonProcessingException {  
 // Arrange  
 EventRequest eventRequest = new EventRequest();  
 String smallCorrelationId = "123"; // A small correlation ID  
 eventRequest.setCorrelationId(smallCorrelationId);  
 EventResponse eventResponse = new EventResponse();  
  
 // Stub the behavior of eventStoreHelper  
 when(eventStoreHelper.getEventDataResponse(eventRequest, eventResponse))  
 .thenReturn(Optional.of(eventResponse));  
  
 // Act  
 Optional<EventResponse> result = DocumentGeneratorEventStoreService.saveEvent(eventRequest);  
  
 // Assert  
 verify(eventStoreHelper, times(1)).getEventDataResponse(eventRequest, eventResponse);  
 assertTrue(result.isPresent());  
 }  
  
 // Helper method to generate a large string  
 private String generateLargeString() {  
 StringBuilder sb = new StringBuilder();  
 for (int i = 0; i < 1000; i++) {  
 sb.append("A"); // Append a large number of characters  
 }  
 return sb.toString();  
 }  
//Method-1 Testcase: 12  
 //Provide incorrect input data to test the service's rejection of invalid or malicious requests  
 @Test  
 public void testSaveEventWithInvalidInput() throws JsonProcessingException {  
 // Arrange  
 EventRequest eventRequest = new EventRequest();  
 eventRequest.setCorrelationId(null); // Provide invalid input, e.g., a null correlationId  
 EventResponse eventResponse = new EventResponse();  
  
 // Stub the behavior of eventStoreHelper to throw DataIntegrityViolationException  
 *when*(eventStoreHelper.getEventDataResponse(eventRequest, eventResponse))  
 .thenThrow(DataIntegrityViolationException.class);  
  
 // Act and Assert  
 *assertThrows*(DataIntegrityViolationException.class, () -> {  
 DocumentGeneratorEventStoreService.*saveEvent*(eventRequest);  
 });  
  
 // Verify that eventStoreHelper was called  
 *verify*(eventStoreHelper, *times*(1)).getEventDataResponse(eventRequest, eventResponse);  
  
 // Verify that eventMapper and responseMapper were never called  
 verify(eventMapper, never()).updateExistingEventResponse(any(), any());  
 verify(responseMapper, never()).mapEventTOEventResponse(any(DmEvent.class), eq(eventResponse));  
 }  
//Method-1 Testcase: 13  
 //Provide incorrect input data to test the service's rejection of invalid or malicious requests  
 @Test  
 public void testSaveEventWithMaliciousInput() throws JsonProcessingException {  
 // Arrange  
 EventRequest eventRequest = new EventRequest();  
 eventRequest.setCorrelationId("'; DROP TABLE events; --"); // Provide malicious input  
 EventResponse eventResponse = new EventResponse();  
  
 // Stub the behavior of eventStoreHelper to throw DataIntegrityViolationException  
 when(eventStoreHelper.getEventDataResponse(eventRequest, eventResponse))  
 .thenThrow(DataIntegrityViolationException.class);  
  
 // Act and Assert  
 assertThrows(DataIntegrityViolationException.class, () -> {  
 DocumentGeneratorEventStoreService.saveEvent(eventRequest);  
 });  
  
 // Verify that eventStoreHelper was called  
 verify(eventStoreHelper, times(1)).getEventDataResponse(eventRequest, eventResponse);  
  
 // Verify that eventMapper and responseMapper were never called  
 verify(eventMapper, never()).updateExistingEventResponse(any(), any());  
 verify(responseMapper, never()).mapEventTOEventResponse(any(DmEvent.class), eq(eventResponse));  
 }  
 //Method-1 Testcase: 13  
 //Perform integration tests to ensure that the service functions correctly as part of an integrated system.  
 @Test  
 public void testIntegrationWithValidInput() throws JsonProcessingException {  
 // Arrange  
 EventRequest eventRequest = new EventRequest();  
 eventRequest.setCorrelationId("validCorrelationId");  
 EventResponse eventResponse = new EventResponse();  
  
 // Stub the behavior of eventStoreHelper  
 when(eventStoreHelper.getEventDataResponse(eventRequest, eventResponse))  
 .thenReturn(Optional.of(eventResponse));  
  
 // Act  
 Optional<EventResponse> result = DocumentGeneratorEventStoreService.saveEvent(eventRequest);  
  
 // Assert  
 verify(eventStoreHelper, times(1)).getEventDataResponse(eventRequest, eventResponse);  
 assertTrue(result.isPresent());  
 }  
  
 //Method-1 Testcase:14 • Simulate long-running operations or timeouts and verify that the service handles them appropriately,  
 // possibly using timeouts in your test framework  
 @Test  
 public void testTimeoutHandling() throws JsonProcessingException {  
 // Arrange  
 EventRequest eventRequest = new EventRequest();  
 eventRequest.setCorrelationId("validCorrelationId");  
 EventResponse eventResponse = new EventResponse();  
  
 // Stub the behavior of eventStoreHelper to simulate a long-running operation  
 when(eventStoreHelper.getEventDataResponse(eventRequest, eventResponse))  
 .thenAnswer(invocation -> {  
 Thread.sleep(5000); // Simulate a 5-second delay  
 return Optional.of(eventResponse);  
 });  
  
 // Act and Assert  
 assertTimeout(Duration.ofSeconds(3), () -> {  
 Optional<EventResponse> result = DocumentGeneratorEventStoreService.saveEvent(eventRequest);  
 assertFalse(result.isPresent()); // Ensure that the result is not present due to timeout  
 });  
  
 // Verify that eventStoreHelper was called  
 verify(eventStoreHelper, times(1)).getEventDataResponse(eventRequest, eventResponse);  
  
 // Verify that eventMapper and responseMapper were never called  
 verify(eventMapper, never()).updateExistingEventResponse(any(), any());  
 verify(responseMapper, never()).mapEventTOEventResponse(any(DmEvent.class), eq(eventResponse));  
 }  
  
 //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* METHOD 2\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
 //1. Positive Scenario - Successful Save:  
 //• Test the method when it is expected to execute successfully.  
 //• Mock the dependencies such as dmEventStatusRepository and eventMapper to return expected values.  
 //• Provide valid input values to eventId and createEventStatusRequest.  
 private DocumentGeneratorEventStoreServiceImpl service;  
 @BeforeEach  
 public void setUp() {  
 MockitoAnnotations.initMocks(this);  
  
 service = new DocumentGeneratorEventStoreServiceImpl();  
 service.dmEventStatusRepository = dmEventStatusRepository;  
 service.eventMapper = eventMapper;  
 service.LOG = logger;  
 }  
 @Test  
 public void testSaveEventStatus\_SuccessfulSave() {  
 // Create a sample eventId and createEventStatusRequest  
 String eventId = "123";  
 CreateEventStatusRequest createEventStatusRequest = createSampleRequest();  
  
 // Create a sample DmEventStatus object  
 DmEventStatus sampleDmEventStatus = createSampleDmEventStatus();  
  
 // Create a sample EventResponse object  
 EventResponse sampleEventResponse = createSampleEventResponse();  
  
 // Mock the behavior of dependencies  
 when(dmEventStatusRepository.save(any(DmEventStatus.class))).thenReturn(sampleDmEventStatus);  
  
 // Mock the behavior of the void method eventMapper.updateEventRequestToEventResponse  
 //To mock a void method, you can use the doNothing() method.  
 doNothing().when(eventMapper).updateEventRequestToEventResponse(any(CreateEventStatusRequest.class), any(EventResponse.class));  
  
  
 // Call the method you want to test  
 Optional<EventResponse> result = service.saveEventStatus(eventId, createEventStatusRequest);  
  
 // Assertions  
 assertEquals(Optional.of(sampleEventResponse), result);  
  
 // Verify that dmEventStatusRepository.save and eventMapper.updateEventRequestToEventResponse were called with expected arguments  
 verify(dmEventStatusRepository, times(1)).save(any(DmEventStatus.class));  
 verify(eventMapper, times(1)).updateEventRequestToEventResponse(eq(createEventStatusRequest), any(EventResponse.class));  
 }  
 // Helper methods to create sample objects  
 private CreateEventStatusRequest createSampleRequest() {  
 // Implement this method to create a sample request  
 CreateEventStatusRequest request = new CreateEventStatusRequest();  
 // Set request properties as needed for the test  
 return request;  
 }  
  
 private DmEventStatus createSampleDmEventStatus() {  
 // Implement this method to create a sample DmEventStatus  
 DmEventStatus dmEventStatus = new DmEventStatus();  
 // Set DmEventStatus properties as needed for the test  
 return dmEventStatus;  
 }  
  
 private EventResponse createSampleEventResponse() {  
 // Implement this method to create a sample EventResponse  
 EventResponse eventResponse = new EventResponse();  
 // Set EventResponse properties as needed for the test  
 return eventResponse;  
 }  
  
  
 //Method-2 Testcase :2  
 //1. Save Event Error Scenario:  
 //• Test the method when an event error request is present in createEventStatusRequest.  
 //• Mock the dependencies such as dmEventStatusRepository and eventMapper to return expected values.  
 //• Ensure that the method correctly calls saveEventError with the event error request.  
  
 @Test  
 public void testSaveEventStatus\_WithEventError() {  
 // Create a sample eventId and createEventStatusRequest with an event error request  
 String eventId = "123";  
 CreateEventStatusRequest createEventStatusRequest = createSampleRequestWithEventError();  
  
 // Create a sample DmEventStatus object  
 DmEventStatus sampleDmEventStatus = createSampleDmEventStatus();  
  
 // Create a sample EventResponse object  
 EventResponse sampleEventResponse = createSampleEventResponse();  
  
 // Mock the behavior of dependencies  
 when(dmEventStatusRepository.save(any(DmEventStatus.class))).thenReturn(sampleDmEventStatus);  
 //when(eventMapper.updateEventRequestToEventResponse(any(CreateEventStatusRequest.class), any(EventResponse.class))).thenReturn(sampleEventResponse);  
 doNothing().when(eventMapper).updateEventRequestToEventResponse(any(CreateEventStatusRequest.class), any(EventResponse.class));  
  
  
 // Call the method you want to test  
 Optional<EventResponse> result = service.saveEventStatus(eventId, createEventStatusRequest);  
  
 // Assertions  
 assertEquals(Optional.of(sampleEventResponse), result);  
  
 // Verify that dmEventStatusRepository.save and eventMapper.updateEventRequestToEventResponse were called with expected arguments  
 verify(dmEventStatusRepository, times(1)).save(any(DmEventStatus.class));  
 verify(eventMapper, times(1)).updateEventRequestToEventResponse(eq(createEventStatusRequest), any(EventResponse.class));  
  
 // Verify that saveEventError was called with the event error request  
 verify(service, times(1)).saveEventError(eq(createEventStatusRequest.getEventStatusRequest().getEventStatus().getEventErrorRequest()), eq(sampleDmEventStatus.getEventId()), eq(sampleDmEventStatus));  
 }  
  
 // Helper methods to create sample objects  
 private CreateEventStatusRequest createSampleRequestWithEventError() {  
 // Implement this method to create a sample request with an event error  
 CreateEventStatusRequest request = new CreateEventStatusRequest();  
// EventStatusRequest.EventStatusDetail eventStatusDetail = new EventStatusRequest.EventStatusDetail();  
  
 // EventStatusRequest.EventErrorRequest eventErrorRequest = new EventStatusRequest.EventErrorRequest();  
 // eventStatusDetail.setEventErrorRequest(eventErrorRequest);  
 //request.setEventStatusRequest(eventStatusDetail);  
 // Set other request properties as needed for the test  
 return request;  
 }  
  
 //Method-2 testcase: 3  
 //1. Save Document Details Scenario:  
 //• Test the method when a document request is present in createEventStatusRequest.  
 //• Mock the dependencies such as dmEventStatusRepository and eventMapper to return expected values.  
 //• Ensure that the method correctly calls saveDocumentDetails with the document request.  
 @Test  
 public void testSaveEventStatus\_WithDocumentRequest() {  
 // Create a sample eventId and createEventStatusRequest with a document request  
 String eventId = "123";  
 CreateEventStatusRequest createEventStatusRequest = createSampleRequestWithDocumentRequest();  
  
 // Create a sample DmEventStatus object  
 DmEventStatus sampleDmEventStatus = createSampleDmEventStatus();  
  
 // Create a sample EventResponse object  
 EventResponse sampleEventResponse = createSampleEventResponse();  
  
 // Mock the behavior of dependencies  
 when(dmEventStatusRepository.save(any(DmEventStatus.class))).thenReturn(sampleDmEventStatus);  
 // when(eventMapper.updateEventRequestToEventResponse(any(CreateEventStatusRequest.class), any(EventResponse.class))).thenReturn(sampleEventResponse);  
  
 doNothing().when(eventMapper).updateEventRequestToEventResponse(any(CreateEventStatusRequest.class), any(EventResponse.class));  
 // Call the method you want to test  
 Optional<EventResponse> result = service.saveEventStatus(eventId, createEventStatusRequest);  
  
 // Assertions  
 assertEquals(Optional.of(sampleEventResponse), result);  
  
 // Verify that dmEventStatusRepository.save and eventMapper.updateEventRequestToEventResponse were called with expected arguments  
 verify(dmEventStatusRepository, times(1)).save(any(DmEventStatus.class));  
 verify(eventMapper, times(1)).updateEventRequestToEventResponse(eq(createEventStatusRequest), any(EventResponse.class));  
  
 // Verify that saveDocumentDetails was called with the document request  
 verify(service, times(1)).saveDocumentDetails(eq(createEventStatusRequest.getEventStatusRequest().getEventStatus().getDocumentRequest()), eq(sampleDmEventStatus.getEventId()), eq(sampleEventResponse));  
 }  
  
 // Helper methods to create sample objects  
 private CreateEventStatusRequest createSampleRequestWithDocumentRequest() {  
 // Implement this method to create a sample request with a document request  
 CreateEventStatusRequest request = new CreateEventStatusRequest();  
// EventStatusRequest.EventStatusDetail eventStatusDetail = new EventStatusRequest.EventStatusDetail();  
// EventStatusRequest.DocumentRequest documentRequest = new EventStatusRequest.DocumentRequest();  
// eventStatusDetail.setDocumentRequest(documentRequest);  
// request.setEventStatusRequest(eventStatusDetail);  
 // Set other request properties as needed for the test  
 return request;  
 }  
//Method 2-testcases -4  
 //Save Event Notify Scenario:  
//• Test the method when an event notify data request is present in createEventStatusRequest.  
//• Mock the dependencies such as dmEventStatusRepository and eventMapper to return expected values.  
//• Ensure that the method correctly calls saveEventNotify with the event notify data request.  
@Test  
public void testSaveEventStatus\_WithEventNotifyDataRequest() {  
 // Create a sample eventId and createEventStatusRequest with an event notify data request  
 String eventId = "123";  
 CreateEventStatusRequest createEventStatusRequest = createSampleRequestWithEventNotifyDataRequest();  
  
 // Create a sample DmEventStatus object  
 DmEventStatus sampleDmEventStatus = createSampleDmEventStatus();  
  
 // Create a sample EventResponse object  
 EventResponse sampleEventResponse = createSampleEventResponse();  
  
 // Mock the behavior of dependencies  
 when(dmEventStatusRepository.save(any(DmEventStatus.class))).thenReturn(sampleDmEventStatus);  
 doNothing().when(eventMapper).updateEventRequestToEventResponse(any(CreateEventStatusRequest.class), any(EventResponse.class));  
 // Mock the behavior of saveEventNotify  
 when(dmEventNotifyRepository.save(any(DmEventNotify.class))).thenReturn(new DmEventNotify());  
  
 // Call the method you want to test  
 Optional<EventResponse> result = service.saveEventStatus(eventId, createEventStatusRequest);  
  
 // Assertions  
 assertEquals(Optional.of(sampleEventResponse), result);  
  
 // Verify that dmEventStatusRepository.save and eventMapper.updateEventRequestToEventResponse were called with expected arguments  
 verify(dmEventStatusRepository, times(1)).save(any(DmEventStatus.class));  
 verify(eventMapper, times(1)).updateEventRequestToEventResponse(eq(createEventStatusRequest), any(EventResponse.class));  
  
 // Verify that saveEventNotify was called with the event notify data request  
 verify(dmEventNotifyRepository, times(1)).save(any(DmEventNotify.class));  
}  
  
 // Helper methods to create sample objects  
 private CreateEventStatusRequest createSampleRequestWithEventNotifyDataRequest() {  
 // Implement this method to create a sample request with an event notify data request  
 CreateEventStatusRequest request = new CreateEventStatusRequest();  
 EventStatusRequest.EventStatusDetail eventStatusDetail = new EventStatusRequest.EventStatusDetail();  
  
// EventStatusRequest.EventNotifyDataRequest eventNotifyDataRequest = new EventStatusRequest.EventNotifyDataRequest();  
// eventStatusDetail.setEventNotifyDataRequest(eventNotifyDataRequest);  
// request.setEventStatusRequest(eventStatusDetail);  
  
 // Set other request properties as needed for the test  
 return request;  
 }  
//Method-2 Testcase:5  
 //Error Handling Scenario:  
 //• Test the method when an exception is thrown, such as a DataAccessException during the repository save operation.  
 //• Mock the dependencies to throw the desired exception.  
 //• Verify that the method handles the exception appropriately, possibly by returning an empty Optional or logging an error.  
@Test  
public void testSaveEventStatus\_ExceptionHandling() {  
 // Create a sample eventId and createEventStatusRequest  
 String eventId = "123";  
 CreateEventStatusRequest createEventStatusRequest = createSampleRequest();  
  
 // Mock the behavior of dependencies to throw DataAccessException when saving  
 when(dmEventStatusRepository.save(any(DmEventStatus.class)))  
 .thenThrow(new DataAccessException("Simulated database error") {});  
  
 // Call the method you want to test  
 Optional<EventResponse> result = service.saveEventStatus(eventId, createEventStatusRequest);  
  
 // Assertions  
 assertTrue(result.isEmpty()); // Check that the result is an empty Optional  
  
 // Verify that dmEventStatusRepository.save was called with expected arguments  
 verify(dmEventStatusRepository, times(1)).save(any(DmEventStatus.class));  
  
 // Verify that eventMapper.updateEventRequestToEventResponse was not called  
 verify(eventMapper, never()).updateEventRequestToEventResponse(any(CreateEventStatusRequest.class), any(EventResponse.class));  
}  
 //Method:2 Testcases -6  
 //• Test the method with null or empty values for eventId and createEventStatusRequest.  
 //• Ensure that the method handles these cases gracefully, possibly returning an empty Optional or logging errors.  
@Test  
public void testSaveEventStatus\_NullEmptyInputs() {  
 // Test with null or empty values for eventId and createEventStatusRequest  
 String nullEventId = null;  
 CreateEventStatusRequest nullRequest = null;  
 String emptyEventId = "";  
 CreateEventStatusRequest emptyRequest = new CreateEventStatusRequest();  
  
 // Call the method with null eventId and request  
 Optional<EventResponse> resultNull = service.saveEventStatus(nullEventId, nullRequest);  
  
 // Call the method with empty eventId and request  
 Optional<EventResponse> resultEmpty = service.saveEventStatus(emptyEventId, emptyRequest);  
  
 // Assertions  
 assertTrue(resultNull.isEmpty()); // Check that the result is an empty Optional for null inputs  
 assertTrue(resultEmpty.isEmpty()); // Check that the result is an empty Optional for empty inputs  
  
 // Verify that dmEventStatusRepository.save and eventMapper.updateEventRequestToEventResponse were not called  
 verify(dmEventStatusRepository, never()).save(any(DmEventStatus.class));  
 verify(eventMapper, never()).updateEventRequestToEventResponse(any(CreateEventStatusRequest.class), any(EventResponse.class));  
}  
 //Method -2 Testcase-7  
 //Concurrency Testing:  
 //Simulate concurrent requests to the method and verify that it maintains data consistency and thread safety.  
 @Test  
 public void testConcurrentSaveEventStatus() throws InterruptedException {  
 int numThreads = 10; // Number of concurrent threads  
 CountDownLatch latch = new CountDownLatch(numThreads);  
 ExecutorService executorService = Executors.newFixedThreadPool(numThreads);  
  
 // Mock the behavior of dmEventStatusRepository.save  
 when(dmEventStatusRepository.save(any(DmEventStatus.class)))  
 .thenAnswer(invocation -> {  
 // Simulate some processing time  
 Thread.sleep(100);  
 return invocation.getArgument(0);  
 });  
  
 for (int i = 0; i < numThreads; i++) {  
 executorService.submit(() -> {  
 try {  
 String eventId = "123";  
 CreateEventStatusRequest createEventStatusRequest = createSampleRequest();  
  
 // Call the method you want to test concurrently  
 service.saveEventStatus(eventId, createEventStatusRequest);  
  
 } catch (Exception e) {  
 e.printStackTrace();  
 } finally {  
 latch.countDown();  
 }  
 });  
 }  
  
 // Wait for all threads to complete  
 latch.await();  
  
 // Verify that dmEventStatusRepository.save was called the expected number of times  
 verify(dmEventStatusRepository, times(numThreads)).save(any(DmEventStatus.class));  
  
 // Clean up  
 executorService.shutdown();  
 }  
 //Method -2 Testcase-8  
 //Timeout Handling:  
 //Simulate long-running operations or timeouts and verify that the method handles them appropriately.  
@Test  
public void testTimeoutHandling1() {  
 // Create a sample eventId and createEventStatusRequest  
 String eventId = "123";  
 CreateEventStatusRequest createEventStatusRequest = createSampleRequest();  
  
 // Mock the behavior of dmEventStatusRepository to throw a timeout exception  
 when(dmEventStatusRepository.save(any(DmEventStatus.class))).thenThrow(new DataAccessException("Timeout") {});  
  
 // Call the method you want to test  
 Optional<EventResponse> result = service.saveEventStatus(eventId, createEventStatusRequest);  
  
 // Assertions  
 assertFalse(result.isPresent()); // Check that the result is empty due to the timeout  
  
 // Verify that dmEventStatusRepository.save was called with expected arguments  
 verify(dmEventStatusRepository, times(1)).save(any(DmEventStatus.class));  
  
 // You can also verify other behaviors, such as logging or error handling, if applicable  
}  
  
}