Class Name: DocumentGeneratorEventStoreServiceImpl

Method: saveEventStatus

```
@Override
@Transactional
public Optional<EventResponse> saveEventStatus(final String eventId,
                                              final CreateEventStatusRequest
createEventStatusRequest) {
   LOG.debug("Inside saveEventStatus method and eventId : {}",
GenericUtil.sanitizeValues(eventId));
   EventResponse eventResponse = new EventResponse();
   Long eventID = Long.valueOf(eventId);

   DmEventStatus dmEventStatus = new DmEventStatus();
   String eventStatus =
createEventStatusRequest.getEventStatusRequest().getEventStatus().getStatus();
   dmEventStatus.setEventId(eventID);
   dmEventStatus.setStatus(eventStatus);
   dmEventStatus = dmEventStatusRepository.save(dmEventStatus);
   setEventStatus(dmEventStatus, eventResponse);
   eventResponse.getEventDataResponse().setEventId(eventId);

   EventStatusRequest.EventStatusDetail eventStatusDetail
           = createEventStatusRequest.getEventStatusRequest().getEventStatus();

   if
(Optional.ofNullable(eventStatusDetail.getEventErrorRequest()).isPresent()) {
       saveEventError(eventStatusDetail.getEventErrorRequest(), eventID,
dmEventStatus);
   }

   if (Optional.ofNullable(eventStatusDetail.getDocumentRequest()).isPresent())
{
       saveDocumentDetails(eventStatusDetail.getDocumentRequest(), eventID,
eventResponse);
   } else if
(Optional.ofNullable(eventStatusDetail.getEventNotifyDataRequest()).isPresent()
) {
       saveEventNotify(eventStatusDetail.getEventNotifyDataRequest(), eventID);
   }

   eventMapper.updateEventRequestToEventResponse(createEventStatusRequest,
eventResponse);
   return Optional.of(eventResponse);
}
```

Below is an explanation of each line of code in the `saveEventStatus` method

Method Signature

```
@Override
@Transactional
public Optional<EventResponse> saveEventStatus(final String eventId,
                                        final CreateEventStatusRequest createEventStatusRequest) {
```

- `**@Override**`: Indicates that this method overrides a method from its parent class or interface.
- `**@Transactional**`: This annotation ensures that the method runs within a transaction, making sure that database operations are atomic.
- `Optional<EventResponse> saveEventStatus(...)`: The method returns an `**Optional**` wrapper around `**EventResponse**`. This means the method could return a `null` value wrapped as an `Optional`.

Logging

```
LOG.debug("Inside saveEventStatus method and eventId : {}",
GenericUtil.sanitizeValues(eventId));
```

- This line logs debugging information. `GenericUtil.sanitizeValues(eventId)` might be cleaning or escaping the `eventId` before logging it.

Initialize EventResponse

```
EventResponse eventResponse = new EventResponse();
```

- Initializes an empty `EventResponse` object, which will be filled with data later.

Parse EventID

```
Long eventID = Long.valueOf(eventId);
```

- Converts the `eventId` string into a `Long` type.

Initialize and Set DmEventStatus

```
DmEventStatus dmEventStatus = new DmEventStatus();
String eventStatus =
createEventStatusRequest.getEventStatusRequest().getEventStatus().getStatus();
dmEventStatus.setEventId(eventID);
dmEventStatus.setStatus(eventStatus);
dmEventStatus = dmEventStatusRepository.save(dmEventStatus);
```

- Initializes a `DmEventStatus` object.
- Extracts `eventStatus` from the `createEventStatusRequest` object.
- Sets `eventID` and `eventStatus` into `dmEventStatus`.
- Saves `dmEventStatus` into the database using the `dmEventStatusRepository` and updates the object to reflect any changes made during the save operation (like generating an ID).

Set Event Status in Response

```
setEventStatus(dmEventStatus, eventResponse);
```

- Calls a method to set the event status information into the `eventResponse` object.

Set EventId in EventDataResponse

```
eventResponse.getEventDataResponse().setEventId(eventId);
```

- Sets the `eventId` in `eventResponse`'s embedded `EventDataResponse`.

Handle Optional Fields

```
EventStatusRequest.EventStatusDetail eventStatusDetail =
createEventStatusRequest.getEventStatusRequest().getEventStatus();
if (Optional.ofNullable(eventStatusDetail.getEventErrorRequest()).isPresent()) {
   saveEventError(eventStatusDetail.getEventErrorRequest(), eventID, dmEventStatus);
}
```

- Checks if `eventErrorRequest` exists; if yes, saves event errors to the database.

```
if (Optional.ofNullable(eventStatusDetail.getDocumentRequest()).isPresent()) {
   saveDocumentDetails(eventStatusDetail.getDocumentRequest(), eventID, eventResponse);
} else if (Optional.ofNullable(eventStatusDetail.getEventNotifyDataRequest()).isPresent()) {
   saveEventNotify(eventStatusDetail.getEventNotifyDataRequest(), eventID);
}
```

- Checks if `documentRequest` exists; if yes, saves document details.
- If `documentRequest` is not present, checks if `eventNotifyDataRequest` exists; if yes, saves event notification details.

Update Event Response Object

```
eventMapper.updateEventRequestToEventResponse(createEventStatusRequest,
eventResponse);
```

- Calls a mapper to update the `eventResponse` object with data from `createEventStatusRequest`.

Return the Response

```
return Optional.of(eventResponse);
```

Wraps `**eventResponse**` in an `**Optional**` and returns it.

Unit Test:

```
import static org.junit.jupiter.api.Assertions.assertEquals;
import static org.mockito.Mockito.*;

@RunWith(MockitoJUnitRunner.class)
public class YourServiceTest {

    @InjectMocks
    private YourServiceImpl yourService; // Assuming the service class is named
`YourServiceImpl`

    @Mock
    private DmEventStatusRepository dmEventStatusRepository;

    @Mock
    private EventMapper eventMapper;

    @Test
    public void testSaveEventStatus() {
        // Arrange
        String eventId = "1";
        Long eventID = Long.valueOf(eventId);
        CreateEventStatusRequest createEventStatusRequest =
mock(CreateEventStatusRequest.class);
        EventStatusRequest.EventStatusDetail eventStatusDetail =
mock(EventStatusRequest.EventStatusDetail.class);

when(createEventStatusRequest.getEventStatusRequest()).thenReturn(eventStatusDetail);
        when(eventStatusDetail.getStatus()).thenReturn("SomeStatus");
        DmEventStatus dmEventStatus = new DmEventStatus();
        dmEventStatus.setEventId(eventID);
        dmEventStatus.setStatus("SomeStatus");

when(dmEventStatusRepository.save(any(DmEventStatus.class))).thenReturn(dmEventStatus);

        // Act
        Optional<EventResponse> result = yourService.saveEventStatus(eventId,
createEventStatusRequest);

        // Assert
        assertEquals(true, result.isPresent()); // Ensure result is present
        assertEquals(eventId, result.get().getEventDataResponse().getEventId()); // Ensure event
ID is set correctly

        // Verifying that save method of the repository was called once
        verify(dmEventStatusRepository, times(1)).save(any(DmEventStatus.class));

        // Additional assertions and verifications can go here based on your specific requirements
    }
}
```

Explanation

Arrange: Here, we're setting up our test data and mocking dependencies. We mock CreateEventStatusRequest and EventStatusDetail to return the data that we expect.

Mocking Repository: We mock the DmEventStatusRepository to return a pre-set DmEventStatus object when its save method is called. This way, we don't interact with the actual database.

Act: We call the method saveEventStatus that we're testing, and capture its return value in result.

Assert: Finally, we verify that the method behaved as expected. We check whether the result is present and whether the returned EventResponse has the expected eventId.

Verification: Optionally, you can verify if specific methods on your mocked dependencies were called. Here, we verify that the repository's save method was called exactly once.

Remember to replace YourServiceImpl and YourServiceTest with the actual names of your service class and test class, respectively.

This is a simplified example. Depending on how complex your actual service method is, you might need to add more setup, more function calls, and more assertions to fully test the method.