Method : 2 API to interact with Database to add status for event

@Operation(summary = "API to interact with Database to add status for event.")  
@ApiResponses(value = {@ApiResponse(responseCode = "200", description = "Event status added successfully",  
 content = {@Content(mediaType = "application/json",  
 schema = @Schema(implementation = EventResponse.class))})})  
@PostMapping(  
 value = DocumentGeneratorEventStoreConstants.*SLASH* + DocumentGeneratorEventStoreConstants.*EVENT\_ENDPOINT* + DocumentGeneratorEventStoreConstants.*SLASH* + DocumentGeneratorEventStoreConstants.*EVENT\_ID\_PATH\_PARAM*,//endpoint-/event/{eventid}  
 consumes = MediaType.*APPLICATION\_JSON\_VALUE*)  
//to control the HTTP status code, headers, and response body that will be returned to the client  
public static ResponseEntity<EventResponse> createEventStatus(@NotNull @PathVariable final String eventId,  
 @Valid @RequestBody final CreateEventStatusRequest  
 createEventStatusRequest) {  
 //correlationId is likely a unique identifier associated with the event status creation request  
 String correlationId = GenericUtil.*sanitizeValues*(createEventStatusRequest.getCorrelationId());  
 //cleaning or validating the value to ensure it meets certain requirements or security standards.  
 *LOG*.info("Document Event data correlation id: {}", correlationId);  
 Optional<EventResponse> responseMetadataOptional =  
 documentGeneratorEventStoreService.saveEventStatus(eventId, createEventStatusRequest);  
 return ResponseEntity.*status*(HttpStatus.*OK*).contentType(MediaType.*APPLICATION\_JSON*)  
 .body(responseMetadataOptional.orElse(null));  
}

**API Endpoint Summary**: This code defines a RESTful API endpoint for adding status information to an event in a document generator application.

**HTTP Method**: This API endpoint is accessible via HTTP POST requests.

**Endpoint URL**: The URL for this endpoint is constructed as follows:

Base Path: **/event**

Path Parameter: **/event/{eventid}** where **{eventid}** is the unique identifier of the event.

**Request Body**:

The request body should be in JSON format.

It should contain a **CreateEventStatusRequest** object that represents the event status information to be added.

The **CreateEventStatusRequest** object undergoes validation based on its annotations.

**Request Path Parameter**:

**eventId**: A required path parameter extracted from the URL. It represents the unique identifier of the event.

**Response**:

HTTP Status Code: 200 (OK) for a successful request.

Content Type: Application JSON

**Response Body**:

The response body will be in JSON format and represents an **EventResponse** object.

If the operation is successful, the **EventResponse** will contain the response metadata.

If the operation fails or the response metadata is not available, **null** will be returned as the response body.

**Logging**:

The code logs the **correlationId** extracted from the request for tracking and debugging purposes.

**Validation and Sanitization**:

The code ensures that the **correlationId** is not null and applies sanitization using the **GenericUtil.sanitizeValues** method.

This API endpoint is used to create and store event status information. It expects a valid JSON request containing event status data and responds with an **EventResponse** in JSON format. Proper validation and sanitation are performed to ensure the security and integrity of the data. The endpoint is also appropriately documented for API consumers.

**Unit Testcases:1**

**Method:2**

//Method: 2 API to interact with Database to add status for event  
  
//There are two tests testCreateEventStatus\_Success,testCreateEventStatus\_NoResponse

@Test  
public void testCreateEventStatus\_Success() throws Exception {  
//setup  
// Create a sample request  
 CreateEventStatusRequest request = new CreateEventStatusRequest();  
 request.setCorrelationId("123");  
  
 // Create a sample response  
 EventResponse response = new EventResponse();  
  
 // Mock the service method to return the response  
 *when*(documentGeneratorEventStoreService.*saveEventStatus*(*anyString*(), *any*(CreateEventStatusRequest.class)))  
 .thenReturn(Optional.*of*(response));  
 /\*any string as the first argument and any instance of CreateEventStatusRequest as the second argument,  
 it should return an Optional containing the response object \*/  
  
 // Perform the POST request and validate the response  
 // Replace with your expected JSON field and value  
 ResultActions expectedVal = mockMvc.perform((RequestBuilder) *post*("/ event/ {eventId}", "event-id")  
 .contentType(MediaType.*APPLICATION\_JSON*)  
 .contentLength(Long.*parseLong*(new ObjectMapper().writeValueAsString(request))))  
 .andExpect(*status*().isOk())  
 .andExpect((ResultMatcher) *content*().contentType(MediaType.*APPLICATION\_JSON*))  
 .andExpect((ResultMatcher) *jsonPath*("$.correlationId").value("123"));  
}

**Test Method Summary:** This JUnit test method is designed to verify the successful execution of the createEventStatus API endpoint by simulating an HTTP POST request and validating the response.

Test Case Description:

**Setup:**

Create a sample CreateEventStatusRequest object named request.

Set the correlationId of the request to "123".

Create a sample EventResponse object named response.

Mock the saveEventStatus method of the documentGeneratorEventStoreService to return an Optional containing the response when provided with any string as the first argument and any instance of CreateEventStatusRequest as the second argument.

**Perform the POST Request:**

Use the mockMvc framework to perform an HTTP POST request to the /event/{eventId} endpoint.

Set the eventId to "event-id".

Set the content type of the request to JSON.

Calculate and set the content length of the request.

Verify that the status code of the response is 200 (OK).

Verify that the content type of the response is JSON.

Verify that the JSON response contains a field named correlationId with a value of "123".

**Note:**

This test method focuses on the successful execution scenario of the createEventStatus endpoint.

The behavior of the saveEventStatus method is mocked to ensure that it returns the response.

The request is constructed, the endpoint is invoked, and the response is validated to ensure that the endpoint behaves correctly.

This test case helps ensure that the createEventStatus API endpoint returns the expected response when provided with a valid request. It validates the correctness of the response status code, content type, and the presence of specific JSON fields in the response.

**UnitTestcase:2**  
 //Method -2 Test case -2 testCreateEventStatusNullEventId  
/\*Test what happens when eventId is null. Ensure that it returns an appropriate error response  
 (e.g., 400 Bad Request).\*/

@Test  
public void testCreateEventStatusNullEventId() {  
//Setup  
 // Arrange: Prepare the test data and behavior  
 String nullEventId = null; // This simulates a null eventId  
 CreateEventStatusRequest request = new CreateEventStatusRequest();  
  
  
 EventResponse response = new EventResponse();  
 response.setCode(HttpStatus.*BAD\_REQUEST*.value());  
 /\*an instance of the EventResponse class is created, and its code property is set to the value of HttpStatus.  
 BAD\_REQUEST.value(),  
 which corresponds to the integer value of the HTTP status code 400 (Bad Request).\*/  
  
  
 // Stub the Mock -the service method to return a response  
 *when*(documentGeneratorEventStoreService.*saveEventStatus*(nullEventId, request))  
 .thenReturn(Optional.*empty*()); // Simulate service returning empty response  
  
 // Act: Perform the actual test  
 ResponseEntity<EventResponse> responseEntity = documentGeneratorEventStoreController.*createEventStatus*(nullEventId, request);  
  
 // Assert: Verify the response and status code  
 *assertNull*(responseEntity);  
 *assertEquals*(HttpStatus.*BAD\_REQUEST*, responseEntity.getStatusCode());  
 /\*The test asserts that the response from the controller has a status code of 400 Bad Request,  
 as expected for the scenario where eventId is null.\*/  
}

**Test Method Summary**: This JUnit test method is designed to test the **createEventStatus** API endpoint when it is provided with a null **eventId**. It verifies the behavior of the controller when an invalid or missing **eventId** is provided.

**Test Case Description**:

**Setup**:

**Arrange**: Prepare the test data and behavior.

Initialize **nullEventId** as **null** to simulate the scenario where **eventId** is null.

Create an empty **CreateEventStatusRequest** object named **request**.

Create an **EventResponse** object named **response** and set its **code** property to the HTTP status code 400 (Bad Request) using **HttpStatus.BAD\_REQUEST.value()**.

**Stub the Mock**: Mock the **saveEventStatus** method of the **documentGeneratorEventStoreService** to return an **Optional** containing no value (**Optional.empty()**) when provided with **nullEventId** and the **request**. This simulates the service returning an empty response.

**Perform the Test**:

**Act**: Perform the actual test.

Invoke the **createEventStatus** method of the **documentGeneratorEventStoreController** with **nullEventId** and **request**, and store the response in **responseEntity**.

**Verify the Test Results**:

**Assert**: Verify the response and status code.

Assert that **responseEntity** is **null**, indicating that no response was returned by the controller.

Assert that the status code of **responseEntity** is **HttpStatus.BAD\_REQUEST**, confirming that the controller correctly handled the null **eventId** scenario and returned a 400 Bad Request status code.

**Note**:

This test method focuses on the scenario where the **eventId** provided to the **createEventStatus** endpoint is null.

It ensures that the controller handles the null **eventId** case by returning an appropriate HTTP status code and no response data.

This test case helps ensure that the **createEventStatus** API endpoint gracefully handles the scenario where the **eventId** is null, responding with a 400 Bad Request status code. It also validates that no response body is returned in this case.

**Unit testcase:3**

//Method 2-testcase -3 Null CreateEventStatusRequest Scenario:  
/\*Test what happens when CreateEventStatusRequest is null.  
Ensure that it returns an appropriate error response (e.g., 400 Bad Request).\*/

@Test  
public void testCreateEventStatusNullRequest() throws Exception {  
 // Arrange: Prepare the test data and behavior  
 String eventId = "sample-event-id";  
 CreateEventStatusRequest nullRequest = null; // Simulate a null request  
  
 // Stub - Mock the service method to return a response  
 *when*(documentGeneratorEventStoreService.*saveEventStatus*(eventId, nullRequest))  
 .thenReturn(Optional.*empty*());  
  
 // Act: Perform the actual test using MockMvc  
 mockMvc.perform((RequestBuilder) *post*("/event/{eventId}", eventId)  
 .contentType(MediaType.*APPLICATION\_JSON*)  
 .contentType(MediaType.*valueOf*("")))  
 // Assert: Verify the response and status code  
 .andExpect(*status*().isOk())  
 .andExpect((ResultMatcher) *content*().string(""));  
 /\*This test case covers the scenario where CreateEventStatusRequest is null and ensures that  
 the controller handles it correctly by returning an empty response.\*/  
}

**Test Method Summary:** This JUnit test method is designed to test the createEventStatus API endpoint when it is provided with a null request (CreateEventStatusRequest). It verifies the behavior of the controller when an invalid or missing request is provided.

**Test Case Description:**

**Setup:**

**Arrange:** Prepare the test data and behavior.

Initialize eventId with the value "sample-event-id" to represent a valid event ID.

Create nullRequest and set it to null to simulate the scenario where the request (CreateEventStatusRequest) is null.

Stub the Mock: Mock the saveEventStatus method of the documentGeneratorEventStoreService to return an Optional containing no value (Optional.empty()) when provided with eventId and the nullRequest. This simulates the service returning an empty response.

**Perform the Test:**

**Act:** Perform the actual test using MockMvc.

Invoke the createEventStatus endpoint with eventId and nullRequest by making an HTTP POST request to the endpoint.

Set the request content type to MediaType.APPLICATION\_JSON.

Set the response content type to an empty MediaType.valueOf("").

**Verify the Test Results:**

**Assert:** Verify the response and status code.

Assert that the response status code is 200 (OK), indicating that the controller correctly handled the null request scenario without errors.

Assert that the response content is an empty string, confirming that the controller returned an empty response body.

**Note:**

This test method focuses on the scenario where the CreateEventStatusRequest is null.

It ensures that the controller handles the null request case by returning an HTTP 200 (OK) status code and an empty response body.

This test case helps ensure that the createEventStatus API endpoint gracefully handles the scenario where the request (CreateEventStatusRequest) is null, responding with a 200 OK status code and no response body.

**Unit Testcase:4**

//Method -2 testcase 4 Empty Correlation ID Scenario:  
/\*• Test what happens when the correlationId within CreateEventStatusRequest is empty.  
 Ensure that it returns an appropriate error response or handles this case gracefully.\*/

@Test  
public void testCreateEventStatusEmptyCorrelationId() throws Exception {  
 //Setup  
 // Arrange: Prepare the test data and behavior  
 String eventId = "sample-event-id";  
 CreateEventStatusRequest request = new CreateEventStatusRequest();  
 request.setCorrelationId(""); // Simulate an empty correlation ID  
  
 // Stub-Mock the service method to return a response  
 *when*(documentGeneratorEventStoreService.*saveEventStatus*(eventId, request))  
 .thenReturn(Optional.*empty*());  
  
 // Act: Perform the actual test using MockMvc  
 mockMvc.perform((RequestBuilder) *post*("/event/{eventId}", eventId)  
 .contentType(MediaType.*APPLICATION\_JSON*)  
 .contentType(MediaType.*valueOf*("")))  
 // Assert: Verify the response and status code  
 .andExpect(*status*().isOk())  
 .andExpect((ResultMatcher) *content*().string(""));  
 /\*it ensures that the controller handles it correctly by returning an empty response.\*/  
}

**Test Method Summary**: This JUnit test method is designed to test the **createEventStatus** API endpoint when it is provided with an empty Correlation ID within the request (**CreateEventStatusRequest**). It verifies the behavior of the controller when an empty Correlation ID is provided.

**Test Case Description**:

**Setup**:

**Arrange**: Prepare the test data and behavior.

Initialize **eventId** with the value **"sample-event-id"** to represent a valid event ID.

Create a **request** of type **CreateEventStatusRequest** and set its Correlation ID to an empty string to simulate an empty Correlation ID scenario.

**Stub the Mock**: Mock the **saveEventStatus** method of the **documentGeneratorEventStoreService** to return an **Optional** containing no value (**Optional.empty()**) when provided with **eventId** and the **request**. This simulates the service returning an empty response.

**Perform the Test**:

**Act**: Perform the actual test using **MockMvc**.

Invoke the **createEventStatus** endpoint with **eventId** and the **request** by making an HTTP POST request to the endpoint.

Set the request content type to **MediaType.APPLICATION\_JSON**.

Set the response content type to an empty **MediaType.valueOf("")**.

**Verify the Test Results**:

**Assert**: Verify the response and status code.

Assert that the response status code is 200 (OK), indicating that the controller correctly handled the scenario of an empty Correlation ID without errors.

Assert that the response content is an empty string, confirming that the controller returned an empty response body.

**Note**:

This test method focuses on the scenario where the Correlation ID within the **CreateEventStatusRequest** is empty.

It ensures that the controller handles the empty Correlation ID case by returning an HTTP 200 (OK) status code and an empty response body.

This test case helps ensure that the **createEventStatus** API endpoint gracefully handles the scenario where the Correlation ID within the request is empty, responding with a 200 OK status code and no response body.

**Unit testcase:5**

//Method--2 testcase -5 EventService Failure Scenario:  
/\* Mock the documentGeneratorEventStoreService to simulate a failure when saving the event status.  
Ensure that the code handles this gracefully and returns an appropriate error response  
 (e.g., 500 Internal Server Error).\*/

@Test  
public void testCreateEventStatusServiceFailure() throws Exception {  
 //Setup  
 // Arrange: Prepare the test data and behavior  
 String eventId = "sample-event-id";  
 CreateEventStatusRequest request = new CreateEventStatusRequest();  
 request.setCorrelationId("123");  
  
 // Mock the service method to throw an exception  
 *when*(documentGeneratorEventStoreService.*saveEventStatus*(eventId, request))  
 .thenThrow(new RuntimeException("Service failed")); // Simulate a service failure  
  
 // Act: Perform the actual test using MockMvc  
 mockMvc.perform((RequestBuilder) *post*("/event/{eventId}", eventId)  
 .contentType(MediaType.*APPLICATION\_JSON*)  
 .contentType(MediaType.*valueOf*("{\"correlationId\":\"123\"}")))  
 // Assert: Verify the response and status code  
 .andExpect(*status*().isInternalServerError())  
 .andExpect((ResultMatcher) *content*().contentType(MediaType.*APPLICATION\_JSON*))  
 .andExpect((ResultMatcher) *jsonPath*("$.message").value("Service failed"));  
}

**Test Method Summary**: This JUnit test method is designed to test the **createEventStatus** API endpoint when an unexpected service failure occurs during the processing of the request. It verifies the behavior of the controller when the underlying service method throws an exception.

**Test Case Description**:

**Setup**:

**Arrange**: Prepare the test data and behavior.

Initialize **eventId** with the value **"sample-event-id"** to represent a valid event ID.

Create a **request** of type **CreateEventStatusRequest** and set its Correlation ID to **"123"** for a valid request.

**Mock the Service**: Mock the **saveEventStatus** method of the **documentGeneratorEventStoreService** to throw a **RuntimeException** with the message **"Service failed"**. This simulates an unexpected service failure scenario.

**Perform the Test**:

**Act**: Perform the actual test using **MockMvc**.

Invoke the **createEventStatus** endpoint with **eventId** and the **request** by making an HTTP POST request to the endpoint.

Set the request content type to **MediaType.APPLICATION\_JSON**.

Set the request body to **MediaType.valueOf("{\"correlationId\":\"123\"}")**.

**Verify the Test Results**:

**Assert**: Verify the response and status code.

Assert that the response status code is 500 (Internal Server Error), indicating that the controller correctly handled the unexpected service failure.

Assert that the response content type is **MediaType.APPLICATION\_JSON**, confirming that the response contains JSON data.

Assert that the JSON response contains a **"message"** field with the value **"Service failed"**, indicating that the exception message from the service has been propagated to the response.

**Note**:

This test method focuses on the scenario where the underlying service method throws a **RuntimeException**, simulating an unexpected service failure.

It ensures that the controller handles such service failures by returning an HTTP 500 (Internal Server Error) status code and an error message in the response.

This test case helps verify that the **createEventStatus** API endpoint handles unexpected service failures gracefully, providing meaningful error responses to the client.

**Unit testcases :6**

//Method-- 2 Testcase - 6 EventService Success Scenario:  
/\*Mock the documentGeneratorEventStoreService to return a successful response.  
Ensure that the code returns a 200 status code with the expected EventResponse.\*/

@Test  
public void testCreateEventStatusServiceSuccess() throws Exception {  
 //Setup  
 // Arrange: Prepare the test data and behavior  
 String eventId = "sample-event-id";  
 CreateEventStatusRequest request = new CreateEventStatusRequest();  
 request.setCorrelationId("123");  
  
 //Stub- Mock the service method to return a successful response  
 EventResponse successfulResponse = new EventResponse();  
 successfulResponse.setCode(HttpStatus.*OK*.value());  
  
 *when*(documentGeneratorEventStoreService.*saveEventStatus*(eventId, request))  
 .thenReturn(Optional.*of*(successfulResponse));  
  
 // Act: Perform the actual test using MockMvc  
 mockMvc.perform((RequestBuilder) (RequestBuilder) *post*("/event/{eventId}", eventId)  
 .contentType(MediaType.*APPLICATION\_JSON*)  
 .contentType(MediaType.*valueOf*("{\"correlationId\":\"123\"}")))  
 // Assert: Verify the response and status code  
 .andExpect(*status*().isOk())  
 .andExpect((ResultMatcher) *content*().contentType(MediaType.*APPLICATION\_JSON*))  
 .andExpect((ResultMatcher) *jsonPath*("$.code").value(HttpStatus.*OK*.value()));  
}

**Test Method Summary:** This JUnit test method is designed to test the createEventStatus API endpoint when the underlying service method succeeds in processing the request. It verifies that the controller returns a successful response when the service operation completes without errors.

**Test Case Description:**

**Setup:**

**Arrange:** Prepare the test data and behavior.

Initialize eventId with the value "sample-event-id" to represent a valid event ID.

Create a request of type CreateEventStatusRequest and set its Correlation ID to "123" for a valid request.

**Stub the Mocked Service:** Mock the saveEventStatus method of the documentGeneratorEventStoreService to return a successful response. Create a successfulResponse object of type EventResponse and set its code to the integer value of HttpStatus.OK.

**Perform the Test:**

**Act:** Perform the actual test using MockMvc.

Invoke the createEventStatus endpoint with eventId and the request by making an HTTP POST request to the endpoint.

Set the request content type to MediaType.APPLICATION\_JSON.

Set the request body to MediaType.valueOf("{\"correlationId\":\"123\"}").

**Verify the Test Results:**

**Assert:** Verify the response and status code.

Assert that the response status code is 200 (OK), indicating that the controller correctly handled the successful service operation.

Assert that the response content type is MediaType.APPLICATION\_JSON, confirming that the response contains JSON data.

Assert that the JSON response contains a "code" field with the value HttpStatus.OK.value(), confirming that the response code matches the expected success status.

**Note:**

This test method focuses on the scenario where the underlying service method completes successfully.

It ensures that the controller correctly returns a successful response (HTTP 200 OK) and that the response contains the expected data.

This test case helps verify that the createEventStatus API endpoint works as expected when the underlying service operation succeeds. It confirms that the controller correctly handles and returns successful responses for valid requests.

**Unit testcase:7**

//Method --2 testcase -7. Optional EventResponse Empty Scenario:  
/\*• Mock the documentGeneratorEventStoreService to return an empty Optional<EventResponse>.  
 Ensure that the code returns a 200 status code with an empty response body\*/

@Test  
public void testCreateEventStatusOptionalEmpty() throws Exception {  
 // Arrange: Prepare the test data and behavior  
 String eventId = "sample-event-id";  
 CreateEventStatusRequest request = new CreateEventStatusRequest();  
 request.setCorrelationId("123");  
  
 // Mock the service method to return an empty Optional  
 *when*(documentGeneratorEventStoreService.*saveEventStatus*(eventId, request))  
 .thenReturn(Optional.*empty*());  
  
 // Act: Perform the actual test using MockMvc  
 mockMvc.perform((RequestBuilder) *post*("/event/{eventId}", eventId)  
 .contentType(MediaType.*APPLICATION\_JSON*)  
 .contentType(MediaType.*valueOf*("{\"correlationId\":\"123\"}")))  
 // Assert: Verify the response and status code  
 .andExpect(*status*().isOk())  
 .andExpect((ResultMatcher) *content*().string(""));  
}

**Test Method Summary**: This JUnit test method is designed to test the **createEventStatus** API endpoint when the underlying service method returns an empty **Optional**. It verifies that the controller correctly handles and responds to an empty response from the service.

**Test Case Description**:

**Setup**:

**Arrange**: Prepare the test data and behavior.

Initialize **eventId** with the value **"sample-event-id"** to represent a valid event ID.

Create a **request** of type **CreateEventStatusRequest** and set its Correlation ID to **"123"** for a valid request.

**Stub the Mocked Service**: Mock the **saveEventStatus** method of the **documentGeneratorEventStoreService** to return an empty **Optional**.

**Perform the Test**:

**Act**: Perform the actual test using **MockMvc**.

Invoke the **createEventStatus** endpoint with **eventId** and the **request** by making an HTTP POST request to the endpoint.

Set the request content type to **MediaType.APPLICATION\_JSON**.

Set the request body to **MediaType.valueOf("{\"correlationId\":\"123\"}")**.

**Verify the Test Results**:

**Assert**: Verify the response and status code.

Assert that the response status code is 200 (OK), indicating that the controller correctly handled the empty response from the service.

Assert that the response content is an empty string (**""**), confirming that the controller returns an empty response body when the service response is empty.

**Note**:

This test method focuses on the scenario where the underlying service method returns an empty **Optional**.

It ensures that the controller correctly handles empty service responses and returns an appropriate empty response to the client.

This test case helps confirm that the **createEventStatus** API endpoint behaves as expected when the underlying service method returns an empty response. It ensures that the controller handles such cases and returns an appropriate response with an empty body.