**Controller Class :** Method 6 (line no 157)

* **Annotations**:

This code uses annotations to define various aspects of the API endpoint:

* **@Operation**: Provides a summary description for the API operation. In this case, it's a description of what this API endpoint does.
* **@ApiResponses**: Specifies the possible responses for this API operation. In this case, it defines a single response with a **200** HTTP status code and a description indicating that "**notify**" details were retrieved successfully. It also specifies that the response will be in **JSON** format and will follow the schema defined by the **EventResponse** class.
* **@GetMapping**: Indicates that this method should handle HTTP GET requests. The URL path for this endpoint is constructed by concatenating several constants defined in the **DocumentGeneratorEventStoreConstants** class, which likely represent parts of the URL path.
* **Method Parameters**:

**final HttpServletRequest httpRequest**: This parameter represents the HTTP request object, which provides context and information about the incoming request. It can be used to access request headers, parameters, and other request-related details.

**@NotNull @RequestParam final String type**: This parameter is annotated with **@RequestParam**, indicating that it is expected to be passed as a query parameter in the URL. The **@NotNul**l annotation enforces that the type parameter must not be null. It represents a type associated with the summary report.

**@NotNull @RequestParam final String startDate**: Similar to the type parameter, this parameter represents the start date for the date range, passed as a query parameter.

**@NotNull @RequestParam final String endDate**: This parameter represents the end date for the date range, also passed as a query parameter.

* **Method Logic**:

**LOG.info(...**): This line logs information about the operation using a logging framework (e.g., Log4j or SLF4J). It includes sanitized versions of the **startDate** and **endDate** parameters, likely for debugging or auditing purposes.

**documentGeneratorEventStoreService.fetchEventSummaryByDateRange(...)**: This line invokes a service method, **fetchEventSummaryByDateRange**, provided by **documentGeneratorEventStoreService**. It passes the sanitized values of **type, startDate,** and **endDate** as arguments. This service method is expected to retrieve event summary data for the specified date range.

**return eventControlResponseMapper(httpRequest, eventDataResponse);**: This line calls the **eventControlResponseMapper** method, which likely processes the retrieved event summary data and maps it to an **EventSummaryResponse** object. The resulting response is wrapped in a **ResponseEntity** and returned to the client.

**Test Case**: 2 test cases for Method 6 :

**1. Success() :** Here’s what the code is doing.

**-public void retrieveSummaryReportByDateRange\_Success()**: This is the method name, which follows the naming convention for JUnit test methods. It suggests that this test case is focused on the successful execution of the **retrieveSummaryReportByDateRange** method.

* **Arrange Phase**:

This section sets up the initial conditions for the test:

**-String type, String startDate, String endDate**: These variables represent sample values for the **type, startDate, and endDate** parameters that will be used as inputs to the **retrieveSummaryReportByDateRange** method.

**String sanitizedType, String sanitizedStartDate, String sanitizedEndDate**: These variables represent the sanitized versions of the input values.

**-MockitoAnnotations.initMocks(this);**: This line initializes the **Mockit**o framework for the current test class, allowing you to use mock objects and behavior verification.

**-when(genericUtil.sanitizeValues(...)).thenReturn(...)**;: These lines mock the behavior of the **genericUti**l object. They specify that when the sanitizeValues method is called with specific input values (e.g., **type, startDate**, or **endDate**), it should return the corresponding sanitized values (e.g., **sanitizedType, sanitizedStartDate**, or **sanitizedEndDate**).

**-when(documentGeneratorEventStoreService.fetchEventSummaryByDateRange(...)).**

**thenReturn(...):** These lines mock the behavior of the **documentGeneratorEventStoreService** by specifying that when the **fetchEventSummaryByDateRange** method is called with specific sanitized input values, it should return an Optional containing an **expectedResponse**, which represents the expected result of the service call.

* **Act Phase:**

**-ResponseEntity<EventSummaryResponse> responseEntity = controller.retrieveSummaryReportByDateRange(...);**: In this line, the **retrieveSummaryReportByDateRange** method of the controller is called with the provided parameters (**httpRequest, type, startDate,** and **endDate**). The result is captured in the **responseEntity** variable.

* **Assert Phase:**

This section contains assertions that verify the correctness of the test's outcome:

**-assertEquals(HttpStatus.OK, responseEntity.getStatusCode());**: This checks whether the HTTP status code of the **responseEntity** is equal to **HttpStatus.OK**, confirming that the response indicates success.

**-assertEquals(expectedResponse, responseEntity.getBody());**: This compares the **expectedResponse** with the response body in the **responseEntity**. It checks if the actual response body matches the expected response, ensuring that the controller produces the expected result.

* **Verification Phase:**

This section verifies that the mocked methods were called with the expected arguments:

**-verify(genericUtil, times(1)).sanitizeValues(...)**;: These lines verify that the **sanitizeValues** method of the **genericUtil** was called exactly once with the specific input values (**type, startDate,** and **endDate**).

**-verify(documentGeneratorEventStoreService, times(1)).fetchEventSummaryByDateRange(...);**: These lines verify that the **fetchEventSummaryByDateRange** method of the **documentGeneratorEventStoreService** was called exactly once with the sanitized input values (**sanitizedType, sanitizedStartDate,** and **sanitizedEndDate**).

======================================================================

**Test Case**: 2 test cases for Method 6 :

**2. Failure() :** Here’s what the code is doing.

**-public void retrieveSummaryReportByDateRange\_Failure():** This is the method name, following the naming convention for JUnit test methods. It suggests that this test case is focused on the failure scenario of the retrieveSummaryReportByDateRange method.

* **Arrange Phase:**

This section sets up the initial conditions for the test, similar to the previous test case:

**String type, String startDate, String endDate:** These variables represent sample values for the **type, startDate,** and **endDate** parameters that will be used as inputs to the **retrieveSummaryReportByDateRange** method.

**String sanitizedType, String sanitizedStartDate, String sanitizedEndDate:** These variables represent the sanitized versions of the input values.

**MockitoAnnotations.initMocks(this);**: This line initializes the Mockito framework for the current test class, allowing you to use mock objects and behavior verification.

**when(genericUtil.sanitizeValues(...)).thenReturn(...);**: These lines mock the behavior of the **genericUtil** object. They specify that when the **sanitizeValues** method is called with specific input values (e.g., **type, startDate,** or **endDate**), it should return the corresponding sanitized values (e.g., **sanitizedType, sanitizedStartDate,** or **sanitizedEndDate**).

**when(documentGeneratorEventStoreService.fetchEventSummaryByDateRange(...)).thenReturn(...):** These lines mock the behavior of the **documentGeneratorEventStoreService**. In this case, they specify that when the **fetchEventSummaryByDateRange** method is called with specific sanitized input values, it should return an empty **Optional**. This simulates a failure scenario where no data is found.

* **Act Phase:**

**-ResponseEntity<EventSummaryResponse> responseEntity = controller.retrieveSummaryReportByDateRange(...);**: In this line, the **retrieveSummaryReportByDateRange** method of the controller is called with the provided parameters (**httpRequest, type, startDate,** and **endDate**). The result is captured in the **responseEntity** variable.

* **Assert Phase:**

This section contains assertions that verify the correctness of the test's outcome:

**-assertEquals(HttpStatus.NOT\_FOUND, responseEntity.getStatusCode());**: This checks whether the HTTP status code of the **responseEntity** is equal to **HttpStatus.NOT\_FOUND**, confirming that the response indicates a failure or resource not found.

**-assertNull(responseEntity.getBody());**: This ensures that the response body is null, indicating that no data was found in the failure scenario.

* **Verification Phase:**

This section verifies that the mocked methods were called with the expected arguments:

**-verify(genericUtil, times(1)).sanitizeValues(...);**: These lines verify that the **sanitizeValues** method of the **genericUtil** was called exactly once with the specific input values (**type, startDate,** and **endDate**).

**-verify(documentGeneratorEventStoreService, times(1)).fetchEventSummaryByDateRange(...);**: These lines verify that the **fetchEventSummaryByDateRange** method of the **documentGeneratorEventStoreService** was called exactly once with the sanitized input values (**sanitizedType, sanitizedStartDate,** and **sanitizedEndDate**).

// Method 6 line no 157 : 2 Test cases  
// 1. Success() API to retrieve summary report by date range  
@Test  
public void retrieveSummaryReportByDateRange\_Success() {  
 // Arrange  
 String type = "sampleType";  
 String startDate = "2023-01-01";  
 String endDate = "2023-12-31";  
 String sanitizedType = "sanitizedType";  
 String sanitizedStartDate = "sanitizedStartDate";  
 String sanitizedEndDate = "sanitizedEndDate";  
  
 // Initialize Mockito  
 MockitoAnnotations.initMocks(this);  
  
 // Mock the behavior of GenericUtil  
 when(genericUtil.sanitizeValues(type)).thenReturn(sanitizedType);  
 when(genericUtil.sanitizeValues(startDate)).thenReturn(sanitizedStartDate);  
 when(genericUtil.sanitizeValues(endDate)).thenReturn(sanitizedEndDate);  
  
 // Mock the behavior of the service  
 EventSummaryResponse expectedResponse = new EventSummaryResponse();  
 when(documentGeneratorEventStoreService.fetchEventSummaryByDateRange(  
 sanitizedType, sanitizedStartDate, sanitizedEndDate))  
 .thenReturn(Optional.of(expectedResponse));  
  
 // Act  
 ResponseEntity<EventSummaryResponse> responseEntity =  
 controller.retrieveSummaryReportByDateRange(httpRequest, type, startDate, endDate);  
  
 // Assert  
 assertEquals(HttpStatus.OK, responseEntity.getStatusCode());  
 assertEquals(expectedResponse, responseEntity.getBody());  
  
 // Verify that GenericUtil.sanitizeValues was called with the correct values  
 verify(genericUtil, times(1)).sanitizeValues(type);  
 verify(genericUtil, times(1)).sanitizeValues(startDate);  
 verify(genericUtil, times(1)).sanitizeValues(endDate);  
  
 // Verify that documentGeneratorEventStoreService.fetchEventSummaryByDateRange was called with sanitized values  
 verify(documentGeneratorEventStoreService, times(1))  
 .fetchEventSummaryByDateRange(sanitizedType, sanitizedStartDate, sanitizedEndDate);  
}

// Method 6 line no 157 : 2 Test cases  
// 2. Failure() API to retrieve summary report by date range  
  
@Test  
public void retrieveSummaryReportByDateRange\_Failure() {  
 // Arrange  
 String type = "sampleType";  
 String startDate = "2023-01-01";  
 String endDate = "2023-12-31";  
 String sanitizedType = "sanitizedType";  
 String sanitizedStartDate = "sanitizedStartDate";  
 String sanitizedEndDate = "sanitizedEndDate";  
  
 // Initialize Mockito  
 MockitoAnnotations.initMocks(this);  
  
 // Mock the behavior of GenericUtil  
 when(genericUtil.sanitizeValues(type)).thenReturn(sanitizedType);  
 when(genericUtil.sanitizeValues(startDate)).thenReturn(sanitizedStartDate);  
 when(genericUtil.sanitizeValues(endDate)).thenReturn(sanitizedEndDate);  
  
 // Mock the behavior of the service to return an empty Optional (failure scenario)  
 when(documentGeneratorEventStoreService.fetchEventSummaryByDateRange(  
 sanitizedType, sanitizedStartDate, sanitizedEndDate))  
 .thenReturn(Optional.empty());  
  
 // Act  
 ResponseEntity<EventSummaryResponse> responseEntity =  
 controller.retrieveSummaryReportByDateRange(httpRequest, type, startDate, endDate);  
  
 // Assert  
 assertEquals(HttpStatus.NOT\_FOUND, responseEntity.getStatusCode());  
 // Ensure the response body is null since no data was found  
 assertNull(responseEntity.getBody());  
  
 // Verify that GenericUtil.sanitizeValues was called with the correct values  
 verify(genericUtil, times(1)).sanitizeValues(type);  
 verify(genericUtil, times(1)).sanitizeValues(startDate);  
 verify(genericUtil, times(1)).sanitizeValues(endDate);  
  
 // Verify that documentGeneratorEventStoreService.fetchEventSummaryByDateRange was called with sanitized values  
 verify(documentGeneratorEventStoreService, times(1))  
 .fetchEventSummaryByDateRange(sanitizedType, sanitizedStartDate, sanitizedEndDate);  
}