**Controller Class :** Method 6 (RetrieveSummaryReportByDateRange)

* **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**: Total 14 test case scenarios for the controller Method 6 :

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

* **Mock Request Parameters:**

**--** The test method begins by defining mock request parameters, including **type, startDate,** and **endDate**. These values are used as inputs to the controller method being tested.

* **Sample Response:**

**--** An instance of **EventSummaryResponse** named **summaryResponse** is created. This represents a mock response that the service being tested will return.

* **Stubbing the Service:**

**--** The when method from the Mockito library is used to stub/mock the behavior of the **documentGeneratorEventStoreService.fetchEventSummaryByDateRange** method. It specifies that when this method is called with the provided parameters (**type, startDate,** and **endDate**), it should return an Optional containing the **summaryResponse**.

* **Execute and Assert:** The code then uses the mockMvc object (presumably set up earlier in the test class) to perform a simulated HTTP GET request to the **/event-summary** endpoint. It includes the mock request parameters as query parameters and specifies the content type as JSON. This is simulating a call to the controller method being tested.

**-- .param("type", type):** Sets the type parameter.

**-- .param("startDate", startDate):** Sets the startDate parameter.

**-- .param("endDate", endDate):** Sets the endDate parameter.

**-- .contentType(MediaType.APPLICATION\_JSON):** Sets the content type of the request.

After making the request, it expects that the HTTP response status code should be **isOk()** (HTTP 200).

* **Verification:**

**--** Finally, the test method uses Mockito's verify to ensure that the **documentGeneratorEventStoreService.fetchEventSummaryByDateRange** method was called exactly once with the expected parameters (**type, startDate,** and **endDate**).

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

**Test Case**: Total 14 test case scenarios for the controller Method 6 :

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

* **Mock Request Parameters:**

**--** The test method begins by defining mock request parameters, including **type, startDate,** and **endDate**. These values are used as inputs to the controller method being tested.

* **Stubbing the Service:**

**--** Using Mockito, the code stubs/mock the behavior of the **documentGeneratorEventStoreService.fetchEventSummaryByDateRange** method. It specifies that when this method is called with the provided parameters (**type, startDate,** and **endDate**), it should return an **Optional** that is empty (**Optional.empty()**). This simulates the scenario where no summary report is found.

* **Execute and Assert:**

**--** Just like the previous test case, the code uses the **mockMvc** object to perform a simulated HTTP GET request to the **/event-summary** endpoint. It includes the mock request parameters as query parameters and specifies the content type as JSON. This is simulating a call to the controller method being tested.

**-- .param("type", type):** Sets the type parameter.

**-- .param("startDate", startDate):** Sets the startDate parameter.

**-- .param("endDate", endDate):** Sets the endDate parameter.

**-- .contentType(MediaType.APPLICATION\_JSON):** Sets the content type of the request.

After making the request, it expects that the HTTP response status code should be **isNotFound()** (**HTTP 404**). This assertion ensures that when no summary report is found, the controller returns a "**not found**" status.

* **Verification**:

**--** Finally, the test method uses Mockito's verify to ensure that the **documentGeneratorEventStoreService.fetchEventSummaryByDateRange** method was called exactly once with the expected parameters (**type, startDate,** and **endDate**).

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

**Test Case**: Total 14 test case scenarios for the controller Method 6 :

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

* **Mock Request Parameters with Invalid Input:**

**--** In this test, the code sets up mock request parameters with **invalid** input. Specifically:

**-- type** is set to **null**, which represents an invalid input because it's expected to have a non-null value. **startDate** and **endDate** are set to valid date strings.

* **Execute and Assert:**

**--** The code uses the mockMvc object to perform a simulated HTTP GET request to the **/event-summary** endpoint, just like in the previous tests. However, this time it includes invalid input (**type** is **null**) as query parameters and specifies the content type as JSON.

**-- .param("type", type):** Sets the type parameter as null.

**-- .param("startDate", startDate):** Sets the startDate parameter.

**-- .param("endDate", endDate):** Sets the endDate parameter.

**-- .contentType(MediaType.APPLICATION\_JSON):** Sets the content type of the request.

After making the request, it expects that the HTTP response status code should be **isBadRequest()** (HTTP **400**). This assertion ensures that when **invalid** input is provided, the controller returns a "**bad request**" status.

* **Verification:**

**--** In this case, the test method uses Mockito's **verifyNoInteractions** method to verify that the **documentGeneratorEventStoreService** was not called at all. This is because the test is intended to check how the controller handles invalid input, and in such cases, the controller should not invoke the service method. **verifyNoInteractions** is used to ensure that there were no interactions with the service.

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

**Test Case**: Total 14 test case scenarios for the controller Method 6 :

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

* **Mock Request Parameters:**

**--** The test method begins by defining mock request parameters, including **type, startDate,** and **endDate**. These values are used as inputs to the controller method being tested.

* **Stubbing the Service:**

**--** Using Mockito, the code stubs/mock the behavior of the **documentGeneratorEventStoreService.fetchEventSummaryByDateRange** method. It specifies that when this method is called with the provided parameters (**type, startDate,** and **endDate**), it should return an empty **Optional**, simulating an internal server error condition.

* **Execute and Assert:**

**--** The code uses the **mockMvc** object to perform a simulated HTTP GET request to the **/event-summary** endpoint, just like in the previous tests. It includes the mock request parameters as query parameters and specifies the content type as JSON.

**-- .param("type", type):** Sets the type parameter.

**-- .param("startDate", startDate):** Sets the startDate parameter.

**-- .param("endDate", endDate):** Sets the endDate parameter.

**-- .contentType(MediaType.APPLICATION\_JSON):** Sets the content type of the request.

After making the request, it expects that the HTTP response status code should be **isInternalServerError()** (HTTP **500**). This assertion ensures that when an internal server error occurs, the controller returns an "**internal server error**" status.

Additionally, it uses andExpect to assert the content type of the response **(MediaType.APPLICATION\_JSON)** and checks the value of the "error" field in the JSON response. It expects the "**error**" field to have the value "**Internal Server Error**".

* **Verification:**

**--** The test method uses Mockito's verify to ensure that the **documentGeneratorEventStoreService.fetchEventSummaryByDateRange** method was called exactly once with the expected parameters (**type, startDate,** and **endDate**).

**Test Case**: Total 14 test case scenarios for the controller Method 6 :

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

* **Mock Request Parameters:**

**--** The test method begins by defining mock request parameters, including **type, startDate,** and **endDate**. These values are used as inputs to the controller method being tested.

**-- type** is set to a valid value.

**-- startDate** is set to "**2023-02-01**," which is a valid date.

**-- endDate** is set to "**2023-01-31**," which is an **invalid** date because it's earlier than the **start date**.

* **Stubbing the Service:**

**--** Using Mockito, the code stubs/mock the behavior of the **documentGeneratorEventStoreService.fetchEventSummaryByDateRange** method. It specifies that when this method is called with the provided parameters (**type, startDate,** and **endDate**), it should return an empty **Optional**, simulating an error condition where an **invalid** date range is provided.

* **Execute and Assert:**

**--** The code uses **MockMvcRequestBuilders** to create a simulated HTTP GET request to the **/event-summary** endpoint, just like in the previous tests. It includes the mock request parameters as query parameters and specifies the content type as JSON.

**-- .param("type", type):** Sets the type parameter.

**-- .param("startDate", startDate):** Sets the startDate parameter.

**-- .param("endDate", endDate):** Sets the endDate parameter.

**-- .contentType(MediaType.APPLICATION\_JSON):** Sets the content type of the request.

After making the request, it expects that the HTTP response status code should be **isBadRequest()** (HTTP **400**). This assertion ensures that when an invalid date range is provided, the controller returns a "**bad request**" status.

* **Verification:**

**--** The test method uses Mockito's verify to ensure that the **documentGeneratorEventStoreService.fetchEventSummaryByDateRange** method was called exactly once with the expected parameters (**type, startDate,** and **endDate**).

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

**Test Case**: Total 14 test case scenarios for the controller Method 6 :

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

* **Mock Request Parameters:**

**--** The test method begins by defining mock request parameters, including **type, startDate,** and **endDate**. These values are used as inputs to the controller method being tested.

* **Stubbing the Service:**

**--** Using Mockito, the code stubs/mock the behavior of the **documentGeneratorEventStoreService.fetchEventSummaryByDateRange** method. It specifies that when this method is called with the provided parameters (**type, startDate,** and **endDate**), it should return an empty **Optional**, simulating a scenario where the service returns an **empty** response.

* **Execute and Assert:**

**--** The code uses mockMvc to perform a simulated HTTP GET request to the **/event-summary** endpoint, just like in the previous tests. It includes the mock request parameters as query parameters and specifies the content type as JSON.

**-- .param("type", type):** Sets the type parameter.

**-- .param("startDate", startDate):** Sets the startDate parameter.

**-- .param("endDate", endDate):** Sets the endDate parameter.

**-- .contentType(MediaType.APPLICATION\_JSON):** Sets the content type of the request.

After making the request, it expects that the HTTP response status code should be **isOk()** (HTTP **200**). This assertion ensures that when the service returns an **empty** response, the controller still responds with a successful status code.

Additionally, it uses **andExpect** to assert that the content of the response body is an **empty** string. This checks that the controller indeed returns an empty response body when the service response is empty.

* **Verification:**

**--** The test method uses Mockito's verify to ensure that the **documentGeneratorEventStoreService.fetchEventSummaryByDateRange** method was called exactly once with the expected parameters (**type, startDate,** and **endDate**).

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

**Test Case**: Total 14 test case scenarios for the controller Method 6 :

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

* **Mock Request Parameters:**

**--** The test method begins by defining mock request parameters, including type, startDate, and endDate. These values are used as inputs to the controller method being tested.

* **Sample EventSummaryResponse:**

**--** An instance of **EventSummaryResponse** named **SummaryResponse** is created. This is used as a sample response for testing. Depending on the actual structure of **EventSummaryResponse**, you would initialize its fields with appropriate data. In this example, it's demonstrated with comments, but you would set properties like **eventId, eventName, eventDate,** and others as needed.

* **Stubbing the Service:**

**--** Using Mockito, the code stubs/mock the behavior of the **documentGeneratorEventStoreService.fetchEventSummaryByDateRange** method. It specifies that when this method is called with the provided parameters (**type, startDate**, and **endDate**), it should return an **Optional** containing the **SummaryResponse**.

* **Execute and Assert / Simulate MVC and Measure Response Time Repeatedly:**

**-- ‘numberOfRequests’** is set to 100, indicating that the test will send 100 requests to the controller.

**-- ‘totalResponseTime’** is initialized to 0, which will be used to calculate the total response time.

Inside a loop that runs 100 times (adjustable as needed), the code does the following:

**--** Records the start time using **System.currentTimeMillis()** before sending the request.

**--** Uses **mockMvc** to perform a simulated HTTP GET request to the **/event-summary** endpoint, just like in the previous tests.

**--** Expects that the HTTP response status code should be **isOk()** (HTTP **200**).

**--** Measures the end time after receiving the response.

**--** Calculates the time taken for the request and adds it to totalResponseTime.

* **Calculate Average Response Time:**

**--** After the loop, the code calculates the average response time by dividing **totalResponseTime** by **numberOfRequests**.

* **Assert Average Response Time:**

**--** The test asserts that the **averageResponseTime** is less than 100 milliseconds. This threshold can be adjusted based on your performance requirements. The assertion checks that the controller's response time is within acceptable performance limits.

* **Verification:**

**--** The test method uses Mockito's verify to ensure that the **documentGeneratorEventStoreService.fetchEventSummaryByDateRange** method was called exactly once with the expected parameters (**type, startDate,** and **endDate**). In this case, it's verified that the service method was called only once for the entire test run.

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

**Test Case**: Total 14 test case scenarios for the controller Method 6 :

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

* **Mock Request Parameters with Empty Dates:**

**--** The test method begins by defining mock request parameters, including **type, startDate,** and **endDate**. In this case, startDate and endDate are set to empty strings, representing missing or invalid date inputs.

**-- type** is set to a valid value.

**-- startDate** and **endDate** are set to **empty** strings, indicating missing date inputs.

* **Stubbing the Service:**

**--** Using Mockito, the code stubs/mock the behavior of the **documentGeneratorEventStoreService.fetchEventSummaryByDateRange** method. It specifies that when this method is called with the provided parameters (**type, startDate,** and **endDate**), it should return an empty **Optional**, simulating a scenario where no summary report is found due to **missing** or **invalid** date inputs.

* **Execute and Assert the MVC Request:**

**--** The code uses **mockMvc** to perform a simulated HTTP GET request to the **/event-summary** endpoint, just like in the previous tests. It includes the mock request parameters as query parameters and specifies the content type as JSON.

**-- .param("type", type):** Sets the type parameter.

**-- .param("startDate", startDate):** Sets the startDate parameter as an empty string.

**-- .param("endDate", endDate):** Sets the endDate parameter as an empty string.

**-- .contentType(MediaType.APPLICATION\_JSON):** Sets the content type of the request.

After making the request, it expects that the HTTP response status code should be **isNotFound()** (HTTP **404**). This assertion ensures that when empty date parameters are provided, the controller returns a "**not found**" status.

* **Verification:**

**--** The test method uses Mockito's **verifyNoInteractions** method to verify that the **documentGeneratorEventStoreService** was not called at all. This is because the test is intended to check how the controller handles empty date parameters, and in such cases, the controller should not invoke the service method.

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

**Test Case**: Total 14 test case scenarios for the controller Method 6 :

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

* **Mock Request Parameters with Null startDate:**

**--** The test method begins by defining mock request parameters, including **type, startDate,** and **endDate**. In this case, **startDate** is set to **null**, representing a missing or invalid input.

**-- type** is set to a valid value.

**-- startDate** is set to **null**, indicating a missing date input.

**-- endDate** is set to a valid date.

* **Execute and Assert the MVC Request with Null startDate:**

**--** The code uses **MockMvcRequestBuilders** to create a simulated HTTP GET request to the **/event-summary** endpoint, just like in the previous tests. It includes the mock request parameters as query parameters and specifies the content type as JSON.

**-- .param("type", type):** Sets the type parameter.

**-- .param("startDate", startDate):** Sets the startDate parameter as null.

**-- .param("endDate", endDate):** Sets the endDate parameter.

**-- .contentType(MediaType.APPLICATION\_JSON):** Sets the content type of the request.

After making the request, it expects that the HTTP response status code should be **isBadRequest()** (HTTP **400**). This assertion ensures that when startDate is set to null, the controller returns a "**bad request**" status.

* **Verification:**

**--** The test method uses Mockito's **verifyNoInteractions** method to verify that the **documentGeneratorEventStoreService** was not called at all. This is because the test is intended to check how the controller handles a missing **startDate**, and in such cases, the controller should not invoke the service method.

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

**Test Case**: Total 14 test case scenarios for the controller Method 6 :

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

* **Mock Request Parameters with Empty startDate:**

**--** The test method begins by defining mock request parameters, including **type, startDate,** and **endDate**. In this case, **startDate** is set to an **empty** string, representing a missing or invalid input.

**-- type** is set to a valid value.

**-- startDate** is set to an **empty** string, indicating an empty date input.

**-- endDate** is set to a valid date.

* **Execute and Assert the MVC Request with Empty startDate:**

**--** The code uses **MockMvcRequestBuilders** to create a simulated HTTP GET request to the **/event-summary** endpoint, just like in the previous tests. It includes the mock request parameters as query parameters and specifies the content type as JSON.

**-- .param("type", type):** Sets the type parameter.

**-- .param("startDate", startDate):** Sets the startDate parameter as an empty string.

**-- .param("endDate", endDate):** Sets the endDate parameter.

**-- .contentType(MediaType.APPLICATION\_JSON):** Sets the content type of the request.

After making the request, it expects that the HTTP response status code should be **isBadRequest()** (HTTP **400**). This assertion ensures that when **startDate** is set to an **empty** string, the controller returns a "**bad request**" status.

* **Verification:**

**--** The test method uses Mockito's **verifyNoInteractions** method to verify that the **documentGeneratorEventStoreService** was not called at all. This is because the test is intended to check how the controller handles an empty **startDate**, and in such cases, the controller should not invoke the service method.

**Test Case**: Total 14 test case scenarios for the controller Method 6 :

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

* **Mock Request Parameters with Null endDate:**

**--** The test method begins by defining mock request parameters, including **type, startDate,** and **endDate**. In this case, endDate is set to null, representing a missing or invalid input.

**-- type** is set to a valid value.

**-- startDate** is set to a valid date.

**-- endDate** is set to **null**, indicating a missing date input.

* **Execute and Assert the MVC Request with Null endDate:**

**--** The code uses **mockMvc** to perform a simulated HTTP GET request to the **/event-summary** endpoint, just like in the previous tests. It includes the mock request parameters as query parameters and specifies the content type as JSON.

**-- .param("type", type):** Sets the type parameter.

**-- .param("startDate", startDate):** Sets the startDate parameter.

**-- .param("endDate", endDate):** Sets the endDate parameter as null.

**-- .contentType(MediaType.APPLICATION\_JSON):** Sets the content type of the request.

After making the request, it expects that the HTTP response status code should be **isBadRequest()** (HTTP **400**). This assertion ensures that when endDate is set to null, the controller returns a "**bad request**" status.

* **Verification:**

**--** The test method uses Mockito's **verifyNoInteractions** method to verify that the **documentGeneratorEventStoreService** was not called at all. This is because the test is intended to check how the controller handles a missing **endDate**, and in such cases, the controller should not invoke the service method.

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

**Test Case**: Total 14 test case scenarios for the controller Method 6 :

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

* **Mock Request Parameters with Empty endDate:**

**--** The test method begins by defining mock request parameters, including **type, startDate,** and **endDate**. In this case, endDate is set to an empty string, representing a missing or invalid input.

**-- type** is set to a valid value.

**-- startDate** is set to a valid date.

**-- endDate** is set to an **empty** string, indicating an empty date input.

* **Execute and Assert the MVC Request with Empty endDate:**

**--** The code uses **mockMvc** to perform a simulated HTTP GET request to the **/event-summary** endpoint, just like in the previous tests. It includes the mock request parameters as query parameters and specifies the content type as JSON.

**-- .param("type", type):** Sets the type parameter.

**-- .param("startDate", startDate):** Sets the startDate parameter.

**-- .param("endDate", endDate):** Sets the endDate parameter as an empty string.

**-- .contentType(MediaType.APPLICATION\_JSON):** Sets the content type of the request.

After making the request, it expects that the HTTP response status code should be **isBadRequest()** (HTTP **400**). This assertion ensures that when **endDate** is set to an empty string, the controller returns a "**bad request**" status.

* **Verification:**

**--** The test method uses Mockito's **verifyNoInteractions** method to verify that the **documentGeneratorEventStoreService** was not called at all. This is because the test is intended to check how the controller handles an empty **endDate**, and in such cases, the controller should not invoke the service method.

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

**Test Case**: Total 14 test case scenarios for the controller Method 6 :

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

* **Mock Request Parameters with All Inputs as null:**

**--** The test method begins by defining mock request parameters, including **type, startDate,** and **endDate**. In this case, all three parameters are set to null, representing missing or invalid inputs.

**-- type** is set to **null**.

**-- startDate** is set to **null**.

**-- endDate** is set to **null**.

* **Execute and Assert the MVC Request with All Inputs as null:**

**--** The code uses **mockMvc** to perform a simulated HTTP GET request to the **/event-summary** endpoint, just like in the previous tests. It includes the mock request parameters as query parameters and specifies the content type as JSON.

**.param("type", type):** Sets the type parameter as null.

**.param("startDate", startDate):** Sets the startDate parameter as null.

**.param("endDate", endDate):** Sets the endDate parameter as null.

**.contentType(MediaType.APPLICATION\_JSON):** Sets the content type of the request.

After making the request, it expects that the HTTP response status code should be **isBadRequest()** (HTTP **400**). This assertion ensures that when all input parameters are set to null, the controller returns a "**bad request**" status.

* **Verification:**

**--** The test method uses Mockito's **verifyNoInteractions** method to verify that the **documentGeneratorEventStoreService** was not called at all. This is because the test is intended to check how the controller handles missing or invalid input parameters, and in such cases, the controller should not invoke the service method.

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

**Test Case**: Total 14 test case scenarios for the controller Method 6 :

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

* **Mock Request Parameters with All Inputs as Empty Strings:**

**--** The test method begins by defining mock request parameters, including **type, startDate,** and **endDate**. In this case, all three parameters are set to empty strings, representing missing or invalid inputs.

**-- type** is set to an empty string.

**-- startDate** is set to an empty string.

**-- endDate** is set to an empty string.

* **Execute and Assert the MVC Request with All Inputs as Empty Strings:**

**--** The code uses **mockMvc** to perform a simulated HTTP GET request to the **/event-summary** endpoint, just like in the previous tests. It includes the mock request parameters as query parameters and specifies the content type as JSON.

**-- .param("type", type):** Sets the type parameter as an empty string.

**-- .param("startDate", startDate):** Sets the startDate parameter as an empty string.

**-- .param("endDate", endDate):** Sets the endDate parameter as an empty string.

**-- .contentType(MediaType.APPLICATION\_JSON):** Sets the content type of the request.

After making the request, it expects that the HTTP response status code should be **isBadRequest()** (HTTP **400**). This assertion ensures that when all input parameters are set to **empty** strings, the controller returns a "**bad request**" status.

* **Verification:**

**--** The test method uses Mockito's **verifyNoInteractions** method to verify that the **documentGeneratorEventStoreService** was not called at all. This is because the test is intended to check how the controller handles missing or invalid input parameters, and in such cases, the controller should not invoke the service method.