@RunWith(SpringRunner.class)  
@WebMvcTest(DocumentGeneratorEventStoreController.class)  
class DocumentGeneratorEventStoreControllerTest {  
  
 @Autowired  
 private MockMvc mockMvc;  
 @MockBean  
 private DocumentGeneratorEventStoreService documentGeneratorEventStoreService;  
 @MockBean  
 private DocumentServiceRequestService documentServiceRequestService;  
  
 @MockBean  
 private GenericUtil genericUtil;  
 //1.)  
@Test  
public void testSaveEventSucess() throws Exception {  
 // setup  
 EventRequest eventRequest = EventRequest.*builder*()  
 .correlationId("correlation-id")  
 .applicationLabel("applicationlabel")  
 .build();  
  
 EventResponse eventResponse = EventResponse.*builder*()  
 .correlationId("correlation-id")  
 .applicationLabel("application-label")  
 .code(Integer.*valueOf*("HttpStatus.CREATED"))  
 .status()  
 .message()  
 .eventDataResponse()  
 .time()  
 .path()  
 .method()  
 .build();  
  
  
 *when*(genericUtil.*sanitizeValues*(*anyString*())).thenReturn("correlation-id");  
 *when*(documentGeneratorEventStoreService.saveEvent(*any*(EventRequest.class))).thenReturn(Optional.*of*(eventResponse));  
  
 // Execute  
  
 mockMvc.perform(MockMvcRequestBuilders.*post*(DocumentGeneratorEventStoreConstants.*SLASH* + DocumentGeneratorEventStoreConstants.*EVENT\_ENDPOINT*)  
 .contentType(MediaType.*APPLICATION\_JSON*)  
 .content(new ObjectMapper().writeValueAsString(eventRequest))  
 .andExpect(*status*().isOk());  
  
 // validate  
 *verify*(documentGeneratorEventStoreService, *times*(1).saveEvent(eventRequest);  
  
}  
  
@Test  
public void testsaveEventExists() throws Exception {  
 // setup data  
 EventRequest eventRequest = EventRequest.*builder*()  
 .correlationId("correlation-id")  
 .applicationLabel("applicationlabel")  
 .build();  
  
 EventResponse eventResponse = EventResponse.*builder*()  
 .correlationId("correlation-id")  
 .applicationLabel("application-label")  
 .code()  
 .status()  
 .message()  
 .eventDataResponse()  
 .time()  
 .path()  
 .method()  
 .build();  
  
  
 *when*(genericUtil.*sanitizeValues*(*anyString*())).thenReturn("correlation-id");  
 *when*(documentGeneratorEventStoreService.saveEvent(*any*(EventRequest.class))).thenReturn(Optional.*of*(eventResponse));  
  
 // Execute  
  
 mockMvc.perform(MockMvcRequestBuilders.*post*(DocumentGeneratorEventStoreConstants.*SLASH* + DocumentGeneratorEventStoreConstants.*EVENT\_ENDPOINT*)  
 .contentType(MediaType.*APPLICATION\_JSON*)  
 .content(new ObjectMapper().writeValueAsString(eventRequest))  
 .andExpect(*status*().isOk());  
  
 // validate  
 *verify*(documentGeneratorEventStoreService, *times*(1).saveEvent(eventRequest));  
  
  
}  
  
@Test  
public void testcreateEventStatus() throws Exception{  
 CreateEventStatusRequest eventStatusRequest = new CreateEventStatusRequest();  
  
 eventStatusRequest.setCorrelationId("correlationId");  
 DmEventStatus dmEventStatus = new DmEventStatus();  
 dmEventStatus.setEventId(1L);  
 EventResponse eventResponse = new EventResponse();  
 eventResponse.setCode(HttpStatus.*OK*.value());  
 String eventId = String.*valueOf*(dmEventStatus.getEventId());  
  
 *when*(documentGeneratorEventStoreService.saveEventStatus(eventId, eventStatusRequest)).thenReturn(Optional.*of*(eventResponse));  
  
  
 mockMvc.perform(*post*(DocumentGeneratorEventStoreConstants.*SLASH* + DocumentGeneratorEventStoreConstants.*EVENT\_ENDPOINT* + DocumentGeneratorEventStoreConstants.*SLASH* + DocumentGeneratorEventStoreConstants.*EVENT\_ID\_PATH\_PARAM*)  
 .contentType(MediaType.*APPLICATION\_JSON*)  
 .content(new ObjectMapper().writeValueAsString(eventStatusRequest)))  
 .andExpect(*status*().isOk());  
  
  
 }  
  
@Test  
public void testretrieveEventByEventId(){  
  
 String eventId = "sampleEventId";  
  
 EventResponse eventResponse = new EventResponse();  
 eventResponse.setCode(HttpStatus.*OK*.value());  
  
 *when*(documentGeneratorEventStoreService.fetchEventByEventId(eventId)).thenReturn(Optional.*of*(eventResponse));  
  
 // execute  
 mockMvc.perform(get("DocumentGeneratorEventStoreConstants.SLASH + DocumentGeneratorEventStoreConstants.EVENT\_ENDPOINT\n" +  
 " + DocumentGeneratorEventStoreConstants.SLASH + DocumentGeneratorEventStoreConstants.EVENT\_ID\_PATH\_PARAM"))  
 .contentType(MediaType.*APPLICATION\_JSON*)  
 .andExpect(*status*().isOk()).andExpect  
}  
  
@Test  
public void testretrieveErrorsResponseByEventId() throws Exception {  
 //setup  
 String eventId = "sample eventId";  
  
 EventResponse eventResponse = new EventResponse();  
 eventResponse.setCode(HttpStatus.*OK*.value());  
  
 *when*(documentGeneratorEventStoreService.fetchErrorsByEventId(eventId)).thenReturn(Optional.*of*(eventResponse));  
  
 // Execute  
 mockMvc.perform(MockMvcRequestBuilders.*get*(DocumentGeneratorEventStoreConstants.*SLASH* + DocumentGeneratorEventStoreConstants.*EVENT\_ENDPOINT* + DocumentGeneratorEventStoreConstants.*SLASH* + DocumentGeneratorEventStoreConstants.*EVENT\_ID\_PATH\_PARAM* + DocumentGeneratorEventStoreConstants.*SLASH* + DocumentGeneratorEventStoreConstants.*ERROR\_ENDPOINT*)  
 .contentType(MediaType.*APPLICATION\_JSON*))  
 .andExpect(*status*().isOk());  
  
}  
@Test  
public void testretrieveDocumentResponseByCustomerAccountUuid() throws Exception {  
 // setup  
 String correlationId = "sample correlation\_Id";  
 String applicationLabel = "sample application\_label";  
 String customerAccountUuid = "sample customerAccountUuid";  
  
 List<InvoiceHistoryDetails> eventResponseHistory = new ArrayList<>();  
  
 *when*(documentGeneratorEventStoreService.fetchResponseByCustomerAccountUuid(customerAccountUuid)).thenReturn(eventResponseHistory);  
 // execute  
  
 mockMvc.perform(MockMvcRequestBuilders.*get*(DocumentGeneratorEventStoreConstants.*GET\_HISTORY\_API\_URL*)  
 .contentType(MediaType.*APPLICATION\_JSON*)  
 .header("sample correlation\_Id")  
 .header("sample application\_label")  
 .pathInfo("sample customerAccountUuid"))  
 .andExpect(*status*().isOk());  
  
  
}  
  
@Test  
public void testretrieveDocumentsByDocumentIds () throws Exception {  
 // create a test data  
 DocumentIdsRequest documentIds = new DocumentIdsRequest();  
  
 // populate document Id's as needed  
 List<DocumentResponse> documentResponseList = new ArrayList<>();  
 // populate document ResponseList as needed  
  
 *when*(documentGeneratorEventStoreService.fetchDocumentsByDocumentIds(*any*(DocumentIdsRequest.class))).thenReturn(documentResponseList);  
  
 // Execute the API call and validate the response  
  
 mockMvc.perform(*post*("/DocumentGeneratorEventStoreConstants.SLASH" +  
 " + DocumentGeneratorEventStoreConstants.INVOICE" +  
 " + DocumentGeneratorEventStoreConstants.SLASH" +  
 " + DocumentGeneratorEventStoreConstants.DOCUMENTS)").contentType(MediaType.*APPLICATION\_JSON*).header("CORRELATION\_ID\_HEADER", "some-correlation-id").header("APPLICATION\_LABEL\_HEADER", "some-application-label").content(new ObjectMapper().writeValueAsString(documentIds))).andExpect(*status*().isOk());  
  
  
 }  
}