package com.project.bos.dg.datastore.controller;  
  
import com.fasterxml.jackson.core.JsonProcessingException;  
import com.fasterxml.jackson.databind.ObjectMapper;  
import com.project.bos.dg.datastore.constants.DocumentGeneratorEventStoreConstants;  
import com.project.bos.dg.datastore.controller.DocumentGeneratorEventStoreController;  
import com.project.bos.dg.datastore.model.entity.DmEventStatus;  
import com.project.bos.dg.datastore.model.request.CreateDocumentServiceRequestStatus;  
import com.project.bos.dg.datastore.model.request.CreateEventStatusRequest;  
import com.project.bos.dg.datastore.model.request.EventRequest;  
import com.project.bos.dg.datastore.model.request.documentbydocumentIds.DocumentIdsRequest;  
import com.project.bos.dg.datastore.model.response.DocumentRetrievalResponse;  
import com.project.bos.dg.datastore.model.response.EventResponse;  
import com.project.bos.dg.datastore.model.response.EventResponseHistory;  
import com.project.bos.dg.datastore.model.response.EventSummaryResponse;  
import com.project.bos.dg.datastore.model.response.common.DocumentResponse;  
import com.project.bos.dg.datastore.model.response.documentservicerequest.DocumentRequestServiceResponse;  
import com.project.bos.dg.datastore.model.response.eventHistory.EventHistoryApiResponse;  
import com.project.bos.dg.datastore.model.response.eventHistory.InvoiceHistoryDetails;  
import com.project.bos.dg.datastore.service.DocumentGeneratorEventStoreService;  
import com.project.bos.dg.datastore.service.DocumentServiceRequestService;  
import com.project.bos.dg.datastore.util.GenericUtil;  
import jakarta.servlet.http.HttpServletRequest;  
  
import org.junit.jupiter.api.Test;  
  
import org.junit.runner.RunWith;  
import org.mockito.ArgumentMatchers;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.boot.test.autoconfigure.web.servlet.WebMvcTest;  
import org.springframework.boot.test.mock.mockito.MockBean;  
import org.springframework.http.HttpStatus;  
import org.springframework.http.MediaType;  
  
import org.springframework.http.ResponseEntity;  
import org.springframework.test.context.junit4.SpringRunner;  
import org.springframework.test.web.servlet.MockMvc;  
  
import org.springframework.test.web.servlet.MvcResult;  
import org.springframework.test.web.servlet.RequestBuilder;  
import org.springframework.test.web.servlet.request.MockMvcRequestBuilders;  
import org.springframework.test.web.servlet.result.MockMvcResultMatchers;  
  
  
import java.util.ArrayList;  
import java.util.List;  
import java.util.Optional;  
import java.util.UUID;  
  
import static jdk.jfr.internal.jfc.model.Constraint.any;  
import static org.hamcrest.Matchers.*hasSize*;  
import static org.hamcrest.Matchers.*nullValue*;  
import static org.junit.jupiter.api.Assertions.\*;  
import static org.mockito.ArgumentMatchers.*anyString*;  
import static org.mockito.Mockito.*mock*;  
import static org.mockito.Mockito.*when*;  
import static org.postgresql.hostchooser.HostRequirement.*any*;  
import static org.springframework.http.RequestEntity.*post*;  
import static org.springframework.test.web.servlet.result.MockMvcResultMatchers.\*;  
import static org.springframework.web.servlet.function.RequestPredicates.*POST*;  
import static org.springframework.web.servlet.function.RequestPredicates.*contentType*;  
  
@RunWith(SpringRunner.class)  
@WebMvcTest(DocumentGeneratorEventStoreController.class)  
public class DocumentGeneratorEventStoreControllerTest {  
 @Autowired  
 private MockMvc mockMvc;  
  
 @MockBean  
 private DocumentGeneratorEventStoreService documentGeneratorEventStoreService;  
 @MockBean  
 private GenericUtil genericUtil;  
  
 @MockBean  
 private HttpServletRequest httpRequest;  
 @Autowired  
 private DocumentGeneratorEventStoreController documentGeneratorEventStoreController;  
 @MockBean  
 private DocumentServiceRequestService documentServiceRequestService;  
  
 @Test  
 public void testCreateDocumentServiceRequestStatus() throws Exception {  
 // Create test data  
 CreateDocumentServiceRequestStatus createDocumentServiceRequestStatus = new CreateDocumentServiceRequestStatus();  
 createDocumentServiceRequestStatus.setApplicationLabel("Some-Application-Label");  
 createDocumentServiceRequestStatus.setCorrelationId("Correlation\_id");  
 UUID documentId = UUID.*randomUUID*();  
  
 DocumentRetrievalResponse documentRetrievalResponse = new DocumentRetrievalResponse();  
 documentRetrievalResponse.setCode(HttpStatus.*CREATED*.value());  
 // stub the mock  
 *when*(documentGeneratorEventStoreService.saveDocumentServiceRequestStatus(documentId,  
 createDocumentServiceRequestStatus)).thenReturn(Optional.*of*(documentRetrievalResponse));  
  
 // Execute  
 mockMvc.perform(MockMvcRequestBuilders.*post*(DocumentGeneratorEventStoreConstants.*SLASH* + DocumentGeneratorEventStoreConstants.*INVOICE* + DocumentGeneratorEventStoreConstants.*SLASH* + DocumentGeneratorEventStoreConstants.*DOCUMENT* + DocumentGeneratorEventStoreConstants.*SLASH* + DocumentGeneratorEventStoreConstants.*DOCUMENT\_ID\_PATH\_PARAM*)  
 .contentType(MediaType.*APPLICATION\_JSON*)  
 .header("CORRELATION\_ID\_HEADER", "Correlation\_id")  
 .header("APPLICATION\_LABEL\_HEADER", "Some-Application-Label")  
 .content(new ObjectMapper().writeValueAsString(createDocumentServiceRequestStatus)))  
 .andExpect(*status*().isCreated())  
 .andExpect(MockMvcResultMatchers.*content*().contentType(MediaType.*APPLICATION\_JSON*));  
 }  
  
 @Test  
 public void testCreateDocumentServiceRequestStatusAlready\_Exists() throws Exception {  
 // Create test data  
 CreateDocumentServiceRequestStatus createDocumentServiceRequestStatus = new CreateDocumentServiceRequestStatus();  
 createDocumentServiceRequestStatus.setApplicationLabel("Some-Application-Label");  
 createDocumentServiceRequestStatus.setCorrelationId("Correlation\_id");  
 UUID documentId = UUID.*randomUUID*();  
  
 DocumentRetrievalResponse documentRetrievalResponse = new DocumentRetrievalResponse();  
 documentRetrievalResponse.setCode(HttpStatus.*OK*.value());  
 *when*(documentGeneratorEventStoreService.saveDocumentServiceRequestStatus(documentId,  
 createDocumentServiceRequestStatus)).thenReturn(Optional.*of*(documentRetrievalResponse));  
  
 // Execute  
 mockMvc.perform(MockMvcRequestBuilders.*post*(DocumentGeneratorEventStoreConstants.*SLASH* + DocumentGeneratorEventStoreConstants.*INVOICE* + DocumentGeneratorEventStoreConstants.*SLASH* + DocumentGeneratorEventStoreConstants.*DOCUMENT* + DocumentGeneratorEventStoreConstants.*SLASH* + DocumentGeneratorEventStoreConstants.*DOCUMENT\_ID\_PATH\_PARAM*)  
 .contentType(MediaType.*APPLICATION\_JSON*)  
 .header("CORRELATION\_ID\_HEADER", "Correlation\_id")  
 .header("APPLICATION\_LABEL\_HEADER", "Some-Application-Label")  
 .content(new ObjectMapper().writeValueAsString(createDocumentServiceRequestStatus)))  
 .andExpect(*status*().isOk())  
 .andExpect(MockMvcResultMatchers.*content*().contentType(MediaType.*APPLICATION\_JSON*));  
 }  
  
 @Test  
 public void testCreateDocumentServiceRequestStatusNo\_Response() throws Exception {  
 // Create test data  
 CreateDocumentServiceRequestStatus createDocumentServiceRequestStatus = new CreateDocumentServiceRequestStatus();  
 createDocumentServiceRequestStatus.setApplicationLabel("Some-Application-Label");  
 createDocumentServiceRequestStatus.setCorrelationId("Correlation\_id");  
 UUID documentId = UUID.*randomUUID*();  
  
  
 *when*(documentGeneratorEventStoreService.saveDocumentServiceRequestStatus(documentId,  
 createDocumentServiceRequestStatus)).thenReturn(Optional.*empty*());  
  
 // Execute  
 mockMvc.perform(MockMvcRequestBuilders.*post*(DocumentGeneratorEventStoreConstants.*SLASH* + DocumentGeneratorEventStoreConstants.*INVOICE* + DocumentGeneratorEventStoreConstants.*SLASH* + DocumentGeneratorEventStoreConstants.*DOCUMENT* + DocumentGeneratorEventStoreConstants.*SLASH* + DocumentGeneratorEventStoreConstants.*DOCUMENT\_ID\_PATH\_PARAM*)  
 .contentType(MediaType.*APPLICATION\_JSON*)  
 .header("CORRELATION\_ID\_HEADER", "Correlation\_id")  
 .header("APPLICATION\_LABEL\_HEADER", "Some-Application-Label")  
 .content(new ObjectMapper().writeValueAsString(createDocumentServiceRequestStatus)))  
 .andExpect(*status*().is(*nullValue*()));  
 }  
  
 @Test  
 public void testSaveEvent() throws Exception {  
 //create test data  
 EventRequest eventRequest = new EventRequest();  
 eventRequest.setCorrelationId("some-Correlation-Id");  
  
 EventResponse eventResponse = new EventResponse();  
 eventResponse.setCode(HttpStatus.*CREATED*.value());  
  
 // stub the mocks  
 *when*(GenericUtil.*sanitizeValues*(*anyString*())).thenReturn("correlationId");  
 *when*(documentGeneratorEventStoreService.saveEvent(eventRequest)).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*().isCreated());  
 }  
  
 @Test  
 public void testRetrieveDocumentDetailsByTypeAndStatus() throws Exception {  
 //create test data  
 DocumentResponse documentResponse = new DocumentResponse();  
 documentResponse.setDocumentStatus("status-pending");  
 documentResponse.setDocumentType("document-type");  
 String documentType = documentResponse.getDocumentType();  
 String documentStatus = documentResponse.getDocumentType();  
 int noOfRecords = 50;  
 List<DocumentRequestServiceResponse> documentRequestServiceResponse = new ArrayList<>();  
 *when*(documentServiceRequestService.  
 fetchByDocumentTypeAndRequestStatus(documentType,  
 documentStatus, noOfRecords)).thenReturn(documentRequestServiceResponse);  
 mockMvc.perform(MockMvcRequestBuilders.*get*(DocumentGeneratorEventStoreConstants.*SLASH* + DocumentGeneratorEventStoreConstants.*DOCUMENT*)  
 .contentType(MediaType.*APPLICATION\_JSON*)  
 .header("CORRELATION\_ID\_HEADER","some-correlation-id")  
 .header("APPLICATION\_LABEL\_HEADER","some Application-label")  
 .param(documentType)  
 .param(documentStatus)  
 .param(String.*valueOf*(noOfRecords))  
 .content(new ObjectMapper().writeValueAsString(documentResponse)))  
 .andExpect(*status*().isOk())  
 .andExpect(*jsonPath*("$", *hasSize*(documentRequestServiceResponse.size())));  
 }  
 @Test  
 public void testRetrieveDocumentResponseByCustomerAccountUuid() throws Exception {  
 EventHistoryApiResponse eventHistoryApiResponse=EventHistoryApiResponse.*builder*()  
 .correlationId("some correlation Id")  
 .applicationLabel("some application label")  
 .build();  
 String customerAccountUuid="some customerAccountUuid";  
 List<InvoiceHistoryDetails> eventResponseHistory =new ArrayList<>();  
  
 *when*(documentGeneratorEventStoreService.fetchResponseByCustomerAccountUuid(customerAccountUuid)).thenReturn(eventResponseHistory);  
  
  
 MvcResult result = mockMvc.perform(MockMvcRequestBuilders.*get*(DocumentGeneratorEventStoreConstants.*GET\_HISTORY\_API\_URL*)  
 .contentType(MediaType.*APPLICATION\_JSON*)  
 .header("CORRELATION\_ID\_HEADER", "some correlation Id")  
 .header("APPLICATION\_LABEL\_HEADER", "some application label")  
 .param(customerAccountUuid)  
 .content(new ObjectMapper().writeValueAsString(eventHistoryApiResponse)))  
 .andReturn();  
 String content = result.getResponse().getContentAsString();  
 ObjectMapper objectMapper=new ObjectMapper();  
 EventResponseHistory actual = objectMapper.readValue(content, EventResponseHistory.class);  
 *assertEquals*(eventResponseHistory,actual.getAdditionalProperties());  
 }  
@Test  
public void testCreateEventStatus() throws Exception {  
 CreateEventStatusRequest createEventStatusRequest=new CreateEventStatusRequest();  
 createEventStatusRequest.setCorrelationId("some-correlation-id");  
 DmEventStatus dmEventStatus=new DmEventStatus();  
 dmEventStatus.setEventId(10L);  
 Long eventId = dmEventStatus.getEventId();  
 EventResponse eventResponse=new EventResponse();  
 eventResponse.setCode(HttpStatus.*OK*.value());  
 *when*(createEventStatusRequest.getCorrelationId()).thenReturn("some\_id");  
 *when*(documentGeneratorEventStoreService.saveEventStatus(String.*valueOf*(eventId), createEventStatusRequest)).thenReturn(Optional.*of*(eventResponse));  
  
 //API call  
 mockMvc.perform(MockMvcRequestBuilders.*post*(DocumentGeneratorEventStoreConstants.*SLASH* + DocumentGeneratorEventStoreConstants.*EVENT\_ENDPOINT* + DocumentGeneratorEventStoreConstants.*SLASH* + DocumentGeneratorEventStoreConstants.*EVENT\_ID\_PATH\_PARAM*).  
 contentType(MediaType.*APPLICATION\_JSON*)  
 .param(String.*valueOf*(eventId))  
 .content(new ObjectMapper().writeValueAsString(createEventStatusRequest)))  
 .andExpect(*status*().isOk());  
}  
 @Test  
 public void testCreateEventStatusNotPresent() throws Exception {  
 CreateEventStatusRequest createEventStatusRequest=new CreateEventStatusRequest();  
 createEventStatusRequest.setCorrelationId("some-correlation-id");  
 DmEventStatus dmEventStatus=new DmEventStatus();  
 dmEventStatus.setEventId(10L);  
 Long eventId = dmEventStatus.getEventId();  
  
 *when*(createEventStatusRequest.getCorrelationId()).thenReturn("some\_id");  
 *when*(documentGeneratorEventStoreService.saveEventStatus(String.*valueOf*(eventId), createEventStatusRequest)).thenReturn(Optional.*empty*());  
  
 //API call  
 mockMvc.perform(MockMvcRequestBuilders.*post*(DocumentGeneratorEventStoreConstants.*SLASH* + DocumentGeneratorEventStoreConstants.*EVENT\_ENDPOINT* + DocumentGeneratorEventStoreConstants.*SLASH* + DocumentGeneratorEventStoreConstants.*EVENT\_ID\_PATH\_PARAM*).  
 contentType(MediaType.*APPLICATION\_JSON*)  
 .param(String.*valueOf*(eventId))  
 .content(new ObjectMapper().writeValueAsString(createEventStatusRequest)))  
 .andExpect(*status*().is(*nullValue*()));  
 }  
 @Test  
 public void testRetrieveDocumentListByEventIdAndStatus() throws Exception {  
 String correlationId="some-id";  
 String applicationLabel="some-application-label";  
 String notStatus="not-status";  
 DocumentRetrievalResponse documentRetrievalResponse=new DocumentRetrievalResponse();  
  
 String eventId="some-eventId";  
 *when*( documentGeneratorEventStoreService.fetchDocumentListByEventIdAndStatus(eventId, notStatus)).thenReturn(Optional.*of*(documentRetrievalResponse));  
  
 //API call  
 MvcResult result = mockMvc.perform(MockMvcRequestBuilders.*get*(DocumentGeneratorEventStoreConstants.*SLASH* + DocumentGeneratorEventStoreConstants.*INVOICE* + DocumentGeneratorEventStoreConstants.*SLASH* + DocumentGeneratorEventStoreConstants.*HISTORY* + DocumentGeneratorEventStoreConstants.*SLASH* + DocumentGeneratorEventStoreConstants.*EVENT\_ID\_PATH\_PARAM*)  
 .contentType(MediaType.*APPLICATION\_JSON*)  
 .header("CORRELATION\_ID\_HEADER", "some-id")  
 .header("APPLICATION\_LABEL\_HEADER", "some-application-label")  
 .param("DocumentGeneratorEventStoreConstants.NOTSTATUS", "nonStatus")  
 .param(eventId, "some-eventId")).andReturn();  
  
 String content = result.getResponse().getContentAsString();  
 ObjectMapper objectMapper=new ObjectMapper();  
 DocumentRetrievalResponse actual = objectMapper.readValue(content,DocumentRetrievalResponse.class);  
 *assertEquals*(documentRetrievalResponse,actual);  
 }  
 @Test  
 public void testRetrieveDocumentsByDocumentIds() throws Exception {  
 // create test data  
 DocumentIdsRequest documentIdsRequest=new DocumentIdsRequest();  
  
 List<DocumentResponse> documentResponses=new ArrayList<>();  
  
 *when*(documentGeneratorEventStoreService  
 .fetchDocumentsByDocumentIds(documentIdsRequest)).thenReturn(documentResponses);  
 mockMvc.perform(MockMvcRequestBuilders.*post*(DocumentGeneratorEventStoreConstants.*SLASH* + DocumentGeneratorEventStoreConstants.*INVOICE* + DocumentGeneratorEventStoreConstants.*SLASH* + DocumentGeneratorEventStoreConstants.*DOCUMENTS*)  
 .header("CORRELATION\_ID\_HEADER","some correlation-id")  
 .header("APPLICATION\_LABEL\_HEADER","some application label")  
 .content(new ObjectMapper().writeValueAsString(documentIdsRequest)))  
 .andExpect(*status*().isOk())  
 .andExpect(*content*().contentType(MediaType.*APPLICATION\_JSON*))  
 .andExpect(*jsonPath*("$",*hasSize*(documentResponses.size())));  
  
 }  
  
 @Test  
 public void testEventControlResponseMapper() {  
 TransactionContext Context = new TransactionContext();  
 httpRequest.setAttribute(BOSConstants.TRANSACTION\_CONTEXT,Context );  
  
 EventSummaryResponse eventSummaryResponse = new EventSummaryResponse();  
 Optional<EventSummaryResponse> eventSummaryResponseOptional= Optional.*of*(new EventSummaryResponse());  
  
 ResponseEntity<EventSummaryResponse> result = documentGeneratorEventStoreController.eventControlResponseMapper(httpRequest, eventSummaryResponseOptional);  
 *assertEquals*(HttpStatus.*OK*,result.getStatusCode());  
 *assertEquals*(MediaType.*APPLICATION\_JSON*, result.getHeaders().getContentType());  
  
 }  
  
 @Test  
 public void testEventControlResponseMapperIsNotPresent() {  
 TransactionContext Context = new TransactionContext();  
 httpRequest.setAttribute(BOSConstants.TRANSACTION\_CONTEXT,Context );  
 Optional<EventSummaryResponse> eventResponse = Optional.*empty*();  
  
 ResponseEntity<EventSummaryResponse> result = documentGeneratorEventStoreController.eventControlResponseMapper(httpRequest, eventResponse);  
  
 *assertEquals*(HttpStatus.*OK*, result.getStatusCode());  
 *assertEquals*(MediaType.*APPLICATION\_JSON*, result.getHeaders().getContentType());  
 *assertNull*(result.getBody());  
  
 }  
  
}