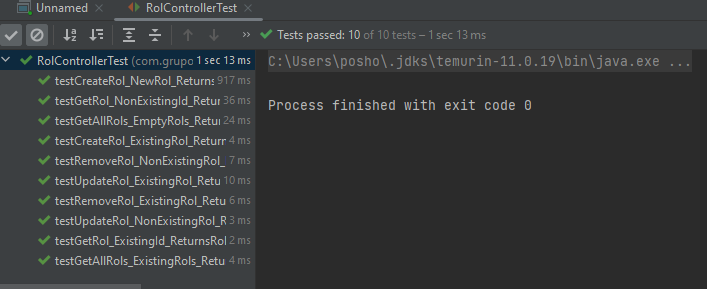
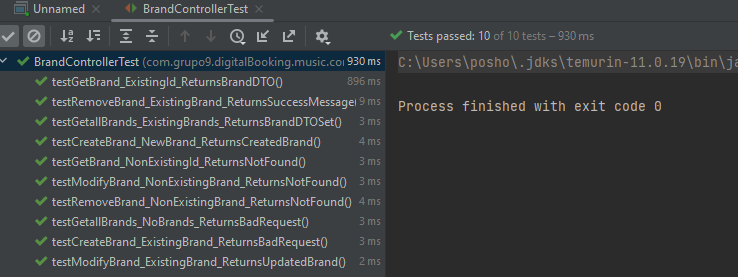
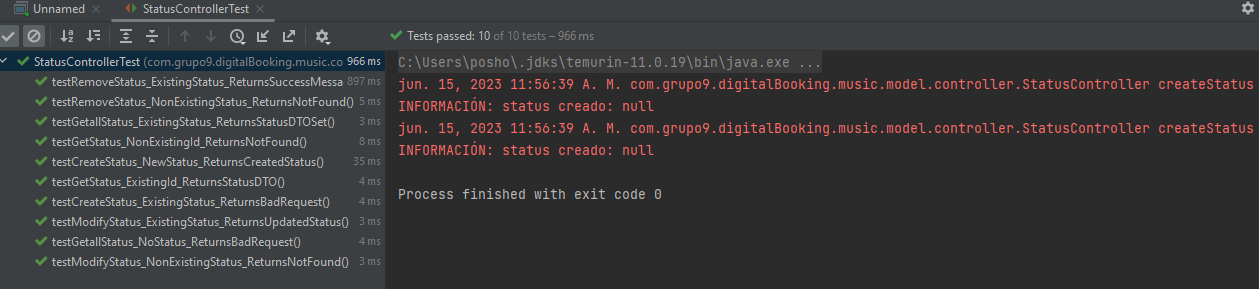
package com.grupo9.digitalBooking.music.controllerTest;  
import static org.junit.jupiter.api.Assertions.*assertEquals*;  
import static org.junit.jupiter.api.Assertions.*assertSame*;  
import static org.mockito.Mockito.\*;  
  
import java.util.HashSet;  
import java.util.Set;  
  
import com.grupo9.digitalBooking.music.model.controller.RolController;  
import org.junit.jupiter.api.BeforeEach;  
import org.junit.jupiter.api.Test;  
import org.mockito.InjectMocks;  
import org.mockito.Mock;  
import org.mockito.MockitoAnnotations;  
import org.springframework.http.HttpStatus;  
import org.springframework.http.ResponseEntity;  
  
import com.grupo9.digitalBooking.music.model.DTO.RolDTO;  
import com.grupo9.digitalBooking.music.model.service.InterfacesService.IRolService;  
  
class RolControllerTest {  
  
 @Mock  
 private IRolService rolService;  
  
 @InjectMocks  
 private RolController rolController;  
  
 @BeforeEach  
 void setUp() {  
 MockitoAnnotations.*openMocks*(this);  
 }  
  
 @Test  
 void testGetRol\_ExistingId\_ReturnsRolDTO() {  
 Long id = 1L;  
 RolDTO rolDTO = new RolDTO();  
 rolDTO.setId(id);  
 rolDTO.setName("Admin");  
  
 *when*(rolService.readRol(id)).thenReturn(rolDTO);  
  
 ResponseEntity<?> response = rolController.getRol(id);  
  
 *assertSame*(HttpStatus.*OK*, response.getStatusCode());  
 *assertSame*(rolDTO, response.getBody());  
  
 *verify*(rolService).readRol(id);  
 *verifyNoMoreInteractions*(rolService);  
 }  
  
 @Test  
 void testGetRol\_NonExistingId\_ReturnsNotFound() {  
 Long id = 1L;  
  
 *when*(rolService.readRol(id)).thenReturn(null);  
  
 ResponseEntity<?> response = rolController.getRol(id);  
  
 *assertSame*(HttpStatus.*BAD\_REQUEST*, response.getStatusCode());  
 *assertEquals*("Message: The role with " + id + " does not exist", response.getBody());  
  
 *verify*(rolService).readRol(id);  
 *verifyNoMoreInteractions*(rolService);  
 }  
  
 @Test  
 void testGetAllRols\_ExistingRols\_ReturnsRolsSet() {  
 Set<RolDTO> rols = new HashSet<>();  
 RolDTO rol1 = new RolDTO();  
 rol1.setId(1L);  
 rol1.setName("Admin");  
 RolDTO rol2 = new RolDTO();  
 rol2.setId(2L);  
 rol2.setName("User");  
 rols.add(rol1);  
 rols.add(rol2);  
  
 *when*(rolService.getAll()).thenReturn(rols);  
  
 ResponseEntity<?> response = rolController.getAllRols();  
  
 *assertSame*(HttpStatus.*OK*, response.getStatusCode());  
 *assertSame*(rols, response.getBody());  
  
 *verify*(rolService).getAll();  
 *verifyNoMoreInteractions*(rolService);  
 }  
  
 @Test  
 void testGetAllRols\_EmptyRols\_ReturnsNotFound() {  
 Set<RolDTO> rols = new HashSet<>();  
  
 *when*(rolService.getAll()).thenReturn(rols);  
  
 ResponseEntity<?> response = rolController.getAllRols();  
  
 *assertSame*(HttpStatus.*BAD\_REQUEST*, response.getStatusCode());  
 *assertEquals*("Message: There are not information", response.getBody());  
  
 *verify*(rolService).getAll();  
 *verifyNoMoreInteractions*(rolService);  
 }  
  
 @Test  
 void testCreateRol\_NewRol\_ReturnsCreatedRol() {  
 RolDTO rolDTO = new RolDTO();  
 rolDTO.setId(1L);  
 rolDTO.setName("Admin");  
  
 *when*(rolService.createRol(rolDTO)).thenReturn(rolDTO);  
  
 ResponseEntity<?> response = rolController.createRol(rolDTO);  
  
 *assertSame*(HttpStatus.*OK*, response.getStatusCode());  
 *assertSame*(rolDTO, response.getBody());  
  
 *verify*(rolService).createRol(rolDTO);  
 *verifyNoMoreInteractions*(rolService);  
 }  
  
 @Test  
 void testCreateRol\_ExistingRol\_ReturnsBadRequest() {  
 RolDTO rolDTO = new RolDTO();  
 rolDTO.setId(1L);  
 rolDTO.setName("Admin");  
  
 *when*(rolService.createRol(rolDTO)).thenReturn(null);  
  
 ResponseEntity<?> response = rolController.createRol(rolDTO);  
  
 *assertSame*(HttpStatus.*BAD\_REQUEST*, response.getStatusCode());  
 *assertEquals*("Message: The role already exists", response.getBody());  
  
 *verify*(rolService).createRol(rolDTO);  
 *verifyNoMoreInteractions*(rolService);  
 }  
  
 @Test  
 void testUpdateRol\_ExistingRol\_ReturnsUpdatedRol() {  
 RolDTO rolDTO = new RolDTO();  
 rolDTO.setId(1L);  
 rolDTO.setName("Admin");  
  
 *when*(rolService.modifyRol(rolDTO)).thenReturn(rolDTO);  
  
 ResponseEntity<?> response = rolController.updateRol(rolDTO);  
  
 *assertSame*(HttpStatus.*OK*, response.getStatusCode());  
 *assertSame*(rolDTO, response.getBody());  
  
 *verify*(rolService).modifyRol(rolDTO);  
 *verifyNoMoreInteractions*(rolService);  
 }  
  
 @Test  
 void testUpdateRol\_NonExistingRol\_ReturnsNotFound() {  
 RolDTO rolDTO = new RolDTO();  
 rolDTO.setId(1L);  
 rolDTO.setName("Admin");  
  
 *when*(rolService.modifyRol(rolDTO)).thenReturn(null);  
  
 ResponseEntity<?> response = rolController.updateRol(rolDTO);  
  
 *assertSame*(HttpStatus.*NOT\_FOUND*, response.getStatusCode());  
 *assertEquals*("Message: The role " + rolDTO.getId() + " does not exist", response.getBody());  
  
 *verify*(rolService).modifyRol(rolDTO);  
 *verifyNoMoreInteractions*(rolService);  
 }  
  
 @Test  
 void testRemoveRol\_ExistingRol\_ReturnsSuccessMessage() {  
 Long id = 1L;  
  
 *when*(rolService.removeRol(id)).thenReturn(true);  
  
 ResponseEntity<?> response = rolController.removeRol(id);  
  
 *assertSame*(HttpStatus.*OK*, response.getStatusCode());  
 *assertEquals*("Message: Role was delete", response.getBody());  
  
 *verify*(rolService).removeRol(id);  
 *verifyNoMoreInteractions*(rolService);  
 }  
  
 @Test  
 void testRemoveRol\_NonExistingRol\_ReturnsNotFound() {  
 Long id = 1L;  
  
 *when*(rolService.removeRol(id)).thenReturn(false);  
  
 ResponseEntity<?> response = rolController.removeRol(id);  
  
 *assertSame*(HttpStatus.*NOT\_FOUND*, response.getStatusCode());  
 *assertEquals*("Message: The role " + id + " does not exist", response.getBody());  
  
 *verify*(rolService).removeRol(id);  
 *verifyNoMoreInteractions*(rolService);  
 }  
}



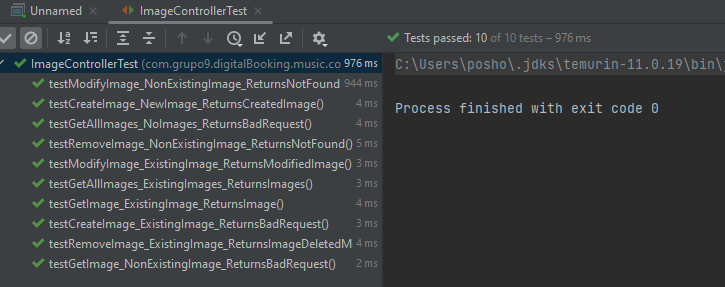
package com.grupo9.digitalBooking.music.controllerTest;  
import static org.junit.jupiter.api.Assertions.*assertEquals*;  
import static org.junit.jupiter.api.Assertions.*assertSame*;  
import static org.mockito.Mockito.\*;  
  
import java.util.HashSet;  
import java.util.Set;  
  
import com.grupo9.digitalBooking.music.model.controller.BrandController;  
import org.junit.jupiter.api.BeforeEach;  
import org.junit.jupiter.api.Test;  
import org.mockito.InjectMocks;  
import org.mockito.Mock;  
import org.mockito.MockitoAnnotations;  
import org.springframework.http.HttpStatus;  
import org.springframework.http.ResponseEntity;  
  
import com.grupo9.digitalBooking.music.model.DTO.BrandDTO;  
import com.grupo9.digitalBooking.music.model.service.InterfacesService.IBrandService;  
  
class BrandControllerTest {  
  
 @Mock  
 private IBrandService brandService;  
  
 @InjectMocks  
 private BrandController brandController;  
  
 @BeforeEach  
 void setUp() {  
 MockitoAnnotations.*openMocks*(this);  
 }  
  
 @Test  
 void testCreateBrand\_NewBrand\_ReturnsCreatedBrand() {  
 BrandDTO brandDTO = new BrandDTO();  
 brandDTO.setId(1L);  
 brandDTO.setName("Brand 1");  
 brandDTO.setImage("brand1.png");  
  
 *when*(brandService.createBrand(brandDTO)).thenReturn(brandDTO);  
  
 ResponseEntity<?> response = brandController.createBrand(brandDTO);  
  
 *assertSame*(HttpStatus.*OK*, response.getStatusCode());  
 *assertSame*(brandDTO, response.getBody());  
  
 *verify*(brandService).createBrand(brandDTO);  
 *verifyNoMoreInteractions*(brandService);  
 }  
  
 @Test  
 void testCreateBrand\_ExistingBrand\_ReturnsBadRequest() {  
 BrandDTO brandDTO = new BrandDTO();  
 brandDTO.setId(1L);  
 brandDTO.setName("Brand 1");  
 brandDTO.setImage("brand1.png");  
  
 *when*(brandService.createBrand(brandDTO)).thenReturn(null);  
  
 ResponseEntity<?> response = brandController.createBrand(brandDTO);  
  
 *assertSame*(HttpStatus.*BAD\_REQUEST*, response.getStatusCode());  
 *assertEquals*("Message: The Brand already exists", response.getBody());  
  
 *verify*(brandService).createBrand(brandDTO);  
 *verifyNoMoreInteractions*(brandService);  
 }  
  
 @Test  
 void testGetBrand\_ExistingId\_ReturnsBrandDTO() {  
 Long id = 1L;  
 BrandDTO brandDTO = new BrandDTO();  
 brandDTO.setId(id);  
 brandDTO.setName("Brand 1");  
 brandDTO.setImage("brand1.png");  
  
 *when*(brandService.readBrand(id)).thenReturn(brandDTO);  
  
 ResponseEntity<?> response = brandController.getBrand(id);  
  
 *assertSame*(HttpStatus.*OK*, response.getStatusCode());  
 *assertSame*(brandDTO, response.getBody());  
  
 *verify*(brandService).readBrand(id);  
 *verifyNoMoreInteractions*(brandService);  
 }  
  
 @Test  
 void testGetBrand\_NonExistingId\_ReturnsNotFound() {  
 Long id = 1L;  
  
 *when*(brandService.readBrand(id)).thenReturn(null);  
  
 ResponseEntity<?> response = brandController.getBrand(id);  
  
 *assertSame*(HttpStatus.*BAD\_REQUEST*, response.getStatusCode());  
 *assertEquals*("Message: The Brand with " + id + " does not exist", response.getBody());  
  
 *verify*(brandService).readBrand(id);  
 *verifyNoMoreInteractions*(brandService);  
 }  
  
 @Test  
 void testModifyBrand\_ExistingBrand\_ReturnsUpdatedBrand() {  
 BrandDTO brandDTO = new BrandDTO();  
 brandDTO.setId(1L);  
 brandDTO.setName("Brand 1");  
 brandDTO.setImage("brand1.png");  
  
 *when*(brandService.modifyBrand(brandDTO)).thenReturn(brandDTO);  
  
 ResponseEntity<?> response = brandController.modifyBrand(brandDTO);  
  
 *assertSame*(HttpStatus.*OK*, response.getStatusCode());  
 *assertSame*(brandDTO, response.getBody());  
  
 *verify*(brandService).modifyBrand(brandDTO);  
 *verifyNoMoreInteractions*(brandService);  
 }  
  
 @Test  
 void testModifyBrand\_NonExistingBrand\_ReturnsNotFound() {  
 BrandDTO brandDTO = new BrandDTO();  
 brandDTO.setId(1L);  
 brandDTO.setName("Brand 1");  
 brandDTO.setImage("brand1.png");  
  
 *when*(brandService.modifyBrand(brandDTO)).thenReturn(null);  
  
 ResponseEntity<?> response = brandController.modifyBrand(brandDTO);  
  
 *assertSame*(HttpStatus.*BAD\_REQUEST*, response.getStatusCode());  
 *assertEquals*("Message: The Brand with " + brandDTO.getId() + " does not exist", response.getBody());  
  
 *verify*(brandService).modifyBrand(brandDTO);  
 *verifyNoMoreInteractions*(brandService);  
 }  
  
 @Test  
 void testRemoveBrand\_ExistingBrand\_ReturnsSuccessMessage() {  
 Long id = 1L;  
  
 *when*(brandService.removeBrand(id)).thenReturn(true);  
  
 ResponseEntity<?> response = brandController.removeBrand(id);  
  
 *assertSame*(HttpStatus.*OK*, response.getStatusCode());  
 *assertEquals*("Message: Brand was delete", response.getBody());  
  
 *verify*(brandService).removeBrand(id);  
 *verifyNoMoreInteractions*(brandService);  
 }  
  
 @Test  
 void testRemoveBrand\_NonExistingBrand\_ReturnsNotFound() {  
 Long id = 1L;  
  
 *when*(brandService.removeBrand(id)).thenReturn(false);  
  
 ResponseEntity<?> response = brandController.removeBrand(id);  
  
 *assertSame*(HttpStatus.*BAD\_REQUEST*, response.getStatusCode());  
 *assertEquals*("Message: The Brand with " + id + " does not exist", response.getBody());  
  
 *verify*(brandService).removeBrand(id);  
 *verifyNoMoreInteractions*(brandService);  
 }  
  
 @Test  
 void testGetallBrands\_ExistingBrands\_ReturnsBrandDTOSet() {  
 Set<BrandDTO> brands = new HashSet<>();  
 BrandDTO brand1 = new BrandDTO();  
 brand1.setId(1L);  
 brand1.setName("Brand 1");  
 brand1.setImage("brand1.png");  
 BrandDTO brand2 = new BrandDTO();  
 brand2.setId(2L);  
 brand2.setName("Brand 2");  
 brand2.setImage("brand2.png");  
 brands.add(brand1);  
 brands.add(brand2);  
  
 *when*(brandService.getAll()).thenReturn(brands);  
  
 ResponseEntity<?> response = brandController.getallBrands();  
  
 *assertSame*(HttpStatus.*OK*, response.getStatusCode());  
 *assertSame*(brands, response.getBody());  
  
 *verify*(brandService).getAll();  
 *verifyNoMoreInteractions*(brandService);  
 }  
  
 @Test  
 void testGetallBrands\_NoBrands\_ReturnsBadRequest() {  
 Set<BrandDTO> brands = new HashSet<>();  
  
 *when*(brandService.getAll()).thenReturn(brands);  
  
 ResponseEntity<?> response = brandController.getallBrands();  
  
 *assertSame*(HttpStatus.*BAD\_REQUEST*, response.getStatusCode());  
 *assertEquals*("Message: There are not information", response.getBody());  
  
 *verify*(brandService).getAll();  
 *verifyNoMoreInteractions*(brandService);  
 }  
}



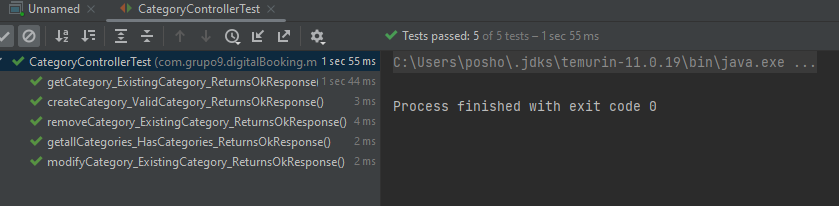
package com.grupo9.digitalBooking.music.controllerTest;  
import static org.junit.jupiter.api.Assertions.*assertEquals*;  
import static org.junit.jupiter.api.Assertions.*assertSame*;  
import static org.mockito.Mockito.\*;  
  
import java.util.HashSet;  
import java.util.Set;  
import java.util.logging.Logger;  
  
import com.grupo9.digitalBooking.music.model.controller.StatusController;  
import org.junit.jupiter.api.BeforeEach;  
import org.junit.jupiter.api.Test;  
import org.mockito.InjectMocks;  
import org.mockito.Mock;  
import org.mockito.MockitoAnnotations;  
import org.springframework.http.HttpStatus;  
import org.springframework.http.ResponseEntity;  
  
import com.grupo9.digitalBooking.music.model.DTO.StatusDTO;  
import com.grupo9.digitalBooking.music.model.service.InterfacesService.IStatusService;  
  
class StatusControllerTest {  
  
 @Mock  
 private IStatusService statusService;  
  
 @InjectMocks  
 private StatusController statusController;  
  
 private static final Logger *LOGGER* = Logger.*getLogger*(String.*valueOf*(StatusController.class));  
  
 @BeforeEach  
 void setUp() {  
 MockitoAnnotations.*openMocks*(this);  
 }  
  
 @Test  
 void testCreateStatus\_NewStatus\_ReturnsCreatedStatus() {  
 StatusDTO statusDTO = new StatusDTO();  
 statusDTO.setId(1L);  
 statusDTO.setName("Status 1");  
  
 *when*(statusService.createStatus(statusDTO)).thenReturn(null);  
  
 ResponseEntity<?> response = statusController.createStatus(statusDTO);  
  
 *assertSame*(HttpStatus.*BAD\_REQUEST*, response.getStatusCode());  
 *assertEquals*("Message: The category already exists", response.getBody());  
  
 *verify*(statusService).createStatus(statusDTO);  
 *verifyNoMoreInteractions*(statusService);  
 }  
  
 @Test  
 void testCreateStatus\_ExistingStatus\_ReturnsBadRequest() {  
 StatusDTO statusDTO = new StatusDTO();  
 statusDTO.setId(1L);  
 statusDTO.setName("Status 1");  
  
 *when*(statusService.createStatus(statusDTO)).thenReturn(null);  
  
 ResponseEntity<?> response = statusController.createStatus(statusDTO);  
  
 *assertSame*(HttpStatus.*BAD\_REQUEST*, response.getStatusCode());  
 *assertEquals*("Message: The category already exists", response.getBody());  
  
 *verify*(statusService).createStatus(statusDTO);  
 *verifyNoMoreInteractions*(statusService);  
 }  
  
 @Test  
 void testGetStatus\_ExistingId\_ReturnsStatusDTO() {  
 Long id = 1L;  
 StatusDTO statusDTO = new StatusDTO();  
 statusDTO.setId(id);  
 statusDTO.setName("Status 1");  
  
 *when*(statusService.readStatus(id)).thenReturn(statusDTO);  
  
 ResponseEntity<?> response = statusController.getStatus(id);  
  
 *assertSame*(HttpStatus.*OK*, response.getStatusCode());  
 *assertSame*(statusDTO, response.getBody());  
  
 *verify*(statusService).readStatus(id);  
 *verifyNoMoreInteractions*(statusService);  
 }  
  
 @Test  
 void testGetStatus\_NonExistingId\_ReturnsNotFound() {  
 Long id = 1L;  
  
 *when*(statusService.readStatus(id)).thenReturn(null);  
  
 ResponseEntity<?> response = statusController.getStatus(id);  
  
 *assertSame*(HttpStatus.*BAD\_REQUEST*, response.getStatusCode());  
 *assertEquals*("Message: The status with " + id + " does not exist", response.getBody());  
  
 *verify*(statusService).readStatus(id);  
 *verifyNoMoreInteractions*(statusService);  
 }  
  
 @Test  
 void testModifyStatus\_ExistingStatus\_ReturnsUpdatedStatus() {  
 StatusDTO statusDTO = new StatusDTO();  
 statusDTO.setId(1L);  
 statusDTO.setName("Status 1");  
  
 *when*(statusService.modifyStatus(statusDTO)).thenReturn(statusDTO);  
  
 ResponseEntity<?> response = statusController.modifyStatus(statusDTO);  
  
 *assertSame*(HttpStatus.*OK*, response.getStatusCode());  
 *assertSame*(statusDTO, response.getBody());  
  
 *verify*(statusService).modifyStatus(statusDTO);  
 *verifyNoMoreInteractions*(statusService);  
 }  
  
 @Test  
 void testModifyStatus\_NonExistingStatus\_ReturnsNotFound() {  
 StatusDTO statusDTO = new StatusDTO();  
 statusDTO.setId(1L);  
 statusDTO.setName("Status 1");  
  
 *when*(statusService.modifyStatus(statusDTO)).thenReturn(null);  
  
 ResponseEntity<?> response = statusController.modifyStatus(statusDTO);  
  
 *assertSame*(HttpStatus.*NOT\_FOUND*, response.getStatusCode());  
 *assertEquals*("Message: The status " + statusDTO.getId() + " does not exist", response.getBody());  
  
 *verify*(statusService).modifyStatus(statusDTO);  
 *verifyNoMoreInteractions*(statusService);  
 }  
  
 @Test  
 void testRemoveStatus\_ExistingStatus\_ReturnsSuccessMessage() {  
 Long id = 1L;  
  
 *when*(statusService.removeStatus(id)).thenReturn(true);  
  
 ResponseEntity<?> response = statusController.removeStatus(id);  
  
 *assertSame*(HttpStatus.*OK*, response.getStatusCode());  
 *assertEquals*("Message: Status was delete", response.getBody());  
  
 *verify*(statusService).removeStatus(id);  
 *verifyNoMoreInteractions*(statusService);  
 }  
  
 @Test  
 void testRemoveStatus\_NonExistingStatus\_ReturnsNotFound() {  
 Long id = 1L;  
  
 *when*(statusService.removeStatus(id)).thenReturn(false);  
  
 ResponseEntity<?> response = statusController.removeStatus(id);  
  
 *assertSame*(HttpStatus.*NOT\_FOUND*, response.getStatusCode());  
 *assertEquals*("Message: The category " + id + " does not exist", response.getBody());  
  
 *verify*(statusService).removeStatus(id);  
 *verifyNoMoreInteractions*(statusService);  
 }  
  
 @Test  
 void testGetallStatus\_ExistingStatus\_ReturnsStatusDTOSet() {  
 Set<StatusDTO> statuses = new HashSet<>();  
 StatusDTO status1 = new StatusDTO();  
 status1.setId(1L);  
 status1.setName("Status 1");  
 StatusDTO status2 = new StatusDTO();  
 status2.setId(2L);  
 status2.setName("Status 2");  
 statuses.add(status1);  
 statuses.add(status2);  
  
 *when*(statusService.getAll()).thenReturn(statuses);  
  
 ResponseEntity<?> response = statusController.getallStatus();  
  
 *assertSame*(HttpStatus.*OK*, response.getStatusCode());  
 *assertSame*(statuses, response.getBody());  
  
 *verify*(statusService).getAll();  
 *verifyNoMoreInteractions*(statusService);  
 }  
  
 @Test  
 void testGetallStatus\_NoStatus\_ReturnsBadRequest() {  
 Set<StatusDTO> statuses = new HashSet<>();  
  
 *when*(statusService.getAll()).thenReturn(statuses);  
  
 ResponseEntity<?> response = statusController.getallStatus();  
  
 *assertSame*(HttpStatus.*BAD\_REQUEST*, response.getStatusCode());  
 *assertEquals*("Message: There are not information", response.getBody());  
  
 *verify*(statusService).getAll();  
 *verifyNoMoreInteractions*(statusService);  
 }  
}



package com.grupo9.digitalBooking.music.controllerTest;  
import static org.junit.jupiter.api.Assertions.\*;  
import static org.mockito.Mockito.\*;  
  
import java.util.HashSet;  
import java.util.Set;  
  
import com.grupo9.digitalBooking.music.model.controller.ImageController;  
import org.junit.jupiter.api.BeforeEach;  
import org.junit.jupiter.api.Test;  
import org.mockito.InjectMocks;  
import org.mockito.Mock;  
import org.mockito.MockitoAnnotations;  
import org.springframework.http.HttpStatus;  
import org.springframework.http.ResponseEntity;  
  
import com.grupo9.digitalBooking.music.model.DTO.ImageDTO;  
import com.grupo9.digitalBooking.music.model.service.InterfacesService.IImageService;  
  
class ImageControllerTest {  
  
 @Mock  
 private IImageService imageService;  
  
 @InjectMocks  
 private ImageController imageController;  
  
 @BeforeEach  
 void setUp() {  
 MockitoAnnotations.*openMocks*(this);  
 }  
  
 @Test  
 void testCreateImage\_NewImage\_ReturnsCreatedImage() {  
 ImageDTO imageDTO = new ImageDTO();  
 imageDTO.setId(1L);  
 imageDTO.setName("Image 1");  
 imageDTO.setUrl("https://example.com/image1.jpg");  
  
 *when*(imageService.createImage(imageDTO)).thenReturn(imageDTO);  
  
 ResponseEntity<?> response = imageController.createImage(imageDTO);  
  
 *assertSame*(HttpStatus.*OK*, response.getStatusCode());  
 *assertSame*(imageDTO, response.getBody());  
  
 *verify*(imageService).createImage(imageDTO);  
 *verifyNoMoreInteractions*(imageService);  
 }  
  
 @Test  
 void testCreateImage\_ExistingImage\_ReturnsBadRequest() {  
 ImageDTO imageDTO = new ImageDTO();  
 imageDTO.setId(1L);  
 imageDTO.setName("Image 1");  
 imageDTO.setUrl("https://example.com/image1.jpg");  
  
 *when*(imageService.createImage(imageDTO)).thenReturn(null);  
  
 ResponseEntity<?> response = imageController.createImage(imageDTO);  
  
 *assertSame*(HttpStatus.*BAD\_REQUEST*, response.getStatusCode());  
 *assertEquals*("Message: The image already exists", response.getBody());  
  
 *verify*(imageService).createImage(imageDTO);  
 *verifyNoMoreInteractions*(imageService);  
 }  
  
 @Test  
 void testGetImage\_ExistingImage\_ReturnsImage() {  
 ImageDTO imageDTO = new ImageDTO();  
 imageDTO.setId(1L);  
 imageDTO.setName("Image 1");  
 imageDTO.setUrl("https://example.com/image1.jpg");  
  
 *when*(imageService.readImage(1L)).thenReturn(imageDTO);  
  
 ResponseEntity<?> response = imageController.getImage(1L);  
  
 *assertSame*(HttpStatus.*OK*, response.getStatusCode());  
 *assertSame*(imageDTO, response.getBody());  
  
 *verify*(imageService).readImage(1L);  
 *verifyNoMoreInteractions*(imageService);  
 }  
  
 @Test  
 void testGetImage\_NonExistingImage\_ReturnsBadRequest() {  
 *when*(imageService.readImage(1L)).thenReturn(null);  
  
 ResponseEntity<?> response = imageController.getImage(1L);  
  
 *assertSame*(HttpStatus.*BAD\_REQUEST*, response.getStatusCode());  
 *assertEquals*("Message: The image with 1 does not exist", response.getBody());  
  
 *verify*(imageService).readImage(1L);  
 *verifyNoMoreInteractions*(imageService);  
 }  
  
 @Test  
 void testModifyImage\_ExistingImage\_ReturnsModifiedImage() {  
 ImageDTO imageDTO = new ImageDTO();  
 imageDTO.setId(1L);  
 imageDTO.setName("Image 1");  
 imageDTO.setUrl("https://example.com/image1.jpg");  
  
 *when*(imageService.modifyImage(imageDTO)).thenReturn(imageDTO);  
  
 ResponseEntity<?> response = imageController.modifyImage(imageDTO);  
  
 *assertSame*(HttpStatus.*OK*, response.getStatusCode());  
 *assertEquals*("Message: Image 1 was update", response.getBody());  
  
 *verify*(imageService).modifyImage(imageDTO);  
 *verifyNoMoreInteractions*(imageService);  
 }  
  
 @Test  
 void testModifyImage\_NonExistingImage\_ReturnsNotFound() {  
 ImageDTO imageDTO = new ImageDTO();  
 imageDTO.setId(1L);  
 imageDTO.setName("Image 1");  
 imageDTO.setUrl("https://example.com/image1.jpg");  
  
 *when*(imageService.modifyImage(imageDTO)).thenReturn(null);  
  
 ResponseEntity<?> response = imageController.modifyImage(imageDTO);  
  
 *assertSame*(HttpStatus.*NOT\_FOUND*, response.getStatusCode());  
 *assertEquals*("Message: The image 1 does not exist", response.getBody());  
  
 *verify*(imageService).modifyImage(imageDTO);  
 *verifyNoMoreInteractions*(imageService);  
 }  
  
 @Test  
 void testRemoveImage\_ExistingImage\_ReturnsImageDeletedMessage() {  
 *when*(imageService.removeImage(1L)).thenReturn(true);  
  
 ResponseEntity<?> response = imageController.removeImage(1L);  
  
 *assertSame*(HttpStatus.*OK*, response.getStatusCode());  
 *assertEquals*("Message: Image was delete", response.getBody());  
  
 *verify*(imageService).removeImage(1L);  
 *verifyNoMoreInteractions*(imageService);  
 }  
  
 @Test  
 void testRemoveImage\_NonExistingImage\_ReturnsNotFound() {  
 *when*(imageService.removeImage(1L)).thenReturn(false);  
  
 ResponseEntity<?> response = imageController.removeImage(1L);  
  
 *assertSame*(HttpStatus.*NOT\_FOUND*, response.getStatusCode());  
 *assertEquals*("Message: The image 1 does not exist", response.getBody());  
  
 *verify*(imageService).removeImage(1L);  
 *verifyNoMoreInteractions*(imageService);  
 }  
  
 @Test  
 void testGetAllImages\_ExistingImages\_ReturnsImages() {  
 Set<ImageDTO> images = new HashSet<>();  
 ImageDTO imageDTO1 = new ImageDTO();  
 imageDTO1.setId(1L);  
 imageDTO1.setName("Image 1");  
 imageDTO1.setUrl("https://example.com/image1.jpg");  
 ImageDTO imageDTO2 = new ImageDTO();  
 imageDTO2.setId(2L);  
 imageDTO2.setName("Image 2");  
 imageDTO2.setUrl("https://example.com/image2.jpg");  
 images.add(imageDTO1);  
 images.add(imageDTO2);  
  
 *when*(imageService.getAll()).thenReturn(images);  
  
 ResponseEntity<?> response = imageController.getallImages();  
  
 *assertSame*(HttpStatus.*OK*, response.getStatusCode());  
 *assertSame*(images, response.getBody());  
  
 *verify*(imageService).getAll();  
 *verifyNoMoreInteractions*(imageService);  
 }  
  
 @Test  
 void testGetAllImages\_NoImages\_ReturnsBadRequest() {  
 Set<ImageDTO> images = new HashSet<>();  
  
 *when*(imageService.getAll()).thenReturn(images);  
  
 ResponseEntity<?> response = imageController.getallImages();  
  
 *assertSame*(HttpStatus.*BAD\_REQUEST*, response.getStatusCode());  
 *assertEquals*("Message: There are not information", response.getBody());  
  
 *verify*(imageService).getAll();  
 *verifyNoMoreInteractions*(imageService);  
 }  
}



package com.grupo9.digitalBooking.music.controllerTest;  
  
import com.grupo9.digitalBooking.music.model.DTO.CategoryDTO;  
import com.grupo9.digitalBooking.music.model.controller.CategoryController;  
import com.grupo9.digitalBooking.music.model.service.InterfacesService.ICategoryService;  
import org.junit.jupiter.api.BeforeEach;  
import org.junit.jupiter.api.Test;  
import org.mockito.InjectMocks;  
import org.mockito.Mock;  
import org.mockito.MockitoAnnotations;  
import org.springframework.http.HttpStatus;  
import org.springframework.http.ResponseEntity;  
  
import java.util.HashSet;  
import java.util.Set;  
  
import static org.junit.jupiter.api.Assertions.*assertEquals*;  
import static org.mockito.Mockito.\*;  
  
class CategoryControllerTest {  
  
 @Mock  
 private ICategoryService categoryService;  
  
 @InjectMocks  
 private CategoryController categoryController;  
  
 @BeforeEach  
 void setUp() {  
 MockitoAnnotations.*openMocks*(this);  
 }  
  
 @Test  
 void createCategory\_ValidCategory\_ReturnsOkResponse() {  
 // Arrange  
 CategoryDTO categoryDTO = new CategoryDTO();  
 categoryDTO.setName("Test Category");  
 categoryDTO.setDescription("Test Description");  
 categoryDTO.setImage("http://example.com/image.jpg");  
  
 *when*(categoryService.createCategory(categoryDTO)).thenReturn(categoryDTO);  
  
 // Act  
 ResponseEntity<?> response = categoryController.createCategory(categoryDTO);  
  
 // Assert  
 *assertEquals*(HttpStatus.*OK*, response.getStatusCode());  
 *assertEquals*(categoryDTO, response.getBody());  
  
 *verify*(categoryService, *times*(1)).createCategory(categoryDTO);  
 }  
  
 @Test  
 void getCategory\_ExistingCategory\_ReturnsOkResponse() {  
 // Arrange  
 Long categoryId = 1L;  
 CategoryDTO categoryDTO = new CategoryDTO();  
 categoryDTO.setId(categoryId);  
 categoryDTO.setName("Test Category");  
 categoryDTO.setDescription("Test Description");  
 categoryDTO.setImage("http://example.com/image.jpg");  
  
 *when*(categoryService.readCategory(categoryId)).thenReturn(categoryDTO);  
  
 // Act  
 ResponseEntity<?> response = categoryController.getCategory(categoryId);  
  
 // Assert  
 *assertEquals*(HttpStatus.*OK*, response.getStatusCode());  
 *assertEquals*(categoryDTO, response.getBody());  
  
 *verify*(categoryService, *times*(1)).readCategory(categoryId);  
 }  
  
 @Test  
 void modifyCategory\_ExistingCategory\_ReturnsOkResponse() {  
 // Arrange  
 CategoryDTO categoryDTO = new CategoryDTO();  
 categoryDTO.setId(1L);  
 categoryDTO.setName("Test Category");  
 categoryDTO.setDescription("Test Description");  
 categoryDTO.setImage("http://example.com/image.jpg");  
  
 *when*(categoryService.modifyCategory(categoryDTO)).thenReturn(categoryDTO);  
  
 // Act  
 ResponseEntity<?> response = categoryController.modifyCategory(categoryDTO);  
  
 // Assert  
 *assertEquals*(HttpStatus.*OK*, response.getStatusCode());  
 *assertEquals*(categoryDTO, response.getBody());  
  
 *verify*(categoryService, *times*(1)).modifyCategory(categoryDTO);  
 }  
  
 @Test  
 void removeCategory\_ExistingCategory\_ReturnsOkResponse() {  
 // Arrange  
 Long categoryId = 1L;  
  
 *when*(categoryService.removeCategory(categoryId)).thenReturn(true);  
  
 // Act  
 ResponseEntity<?> response = categoryController.removeCategory(categoryId);  
  
 // Assert  
 *assertEquals*(HttpStatus.*OK*, response.getStatusCode());  
 *assertEquals*("Message: Category was delete", response.getBody());  
  
 *verify*(categoryService, *times*(1)).removeCategory(categoryId);  
 }  
  
 @Test  
 void getallCategories\_HasCategories\_ReturnsOkResponse() {  
 // Arrange  
 CategoryDTO categoryDTO1 = new CategoryDTO();  
 categoryDTO1.setId(1L);  
 categoryDTO1.setName("Category 1");  
 categoryDTO1.setDescription("Description 1");  
 categoryDTO1.setImage("http://example1.com/image1.jpg");  
  
 CategoryDTO categoryDTO2 = new CategoryDTO();  
 categoryDTO2.setId(2L);  
 categoryDTO2.setName("Category 2");  
 categoryDTO2.setDescription("Description 2");  
 categoryDTO2.setImage("http://example2.com/image2.jpg");  
  
 Set<CategoryDTO> categories = new HashSet<>();  
 categories.add(categoryDTO1);  
 categories.add(categoryDTO2);  
  
 *when*(categoryService.getAll()).thenReturn(categories);  
  
 // Act  
 ResponseEntity<?> response = categoryController.getallCategories();  
  
 // Assert  
 *assertEquals*(HttpStatus.*OK*, response.getStatusCode());  
 *assertEquals*(categories, response.getBody());  
  
 *verify*(categoryService, *times*(1)).getAll();  
 }  
}



package com.grupo9.digitalBooking.music.controllerTest;

import com.grupo9.digitalBooking.music.model.DTO.InstrumentDetailDTO;  
import com.grupo9.digitalBooking.music.model.controller.InstrumentDetailController;  
import com.grupo9.digitalBooking.music.model.service.InterfacesService.IInstrumentDetailService;  
import org.junit.jupiter.api.BeforeEach;  
import org.junit.jupiter.api.Test;  
import org.mockito.InjectMocks;  
import org.mockito.Mock;  
import org.mockito.MockitoAnnotations;  
import org.springframework.http.HttpStatus;  
import org.springframework.http.ResponseEntity;  
  
import java.util.HashSet;  
import java.util.Set;  
  
import static org.junit.jupiter.api.Assertions.*assertEquals*;  
import static org.mockito.Mockito.\*;  
  
public class InstrumentDetailControllerTest {  
 @Mock  
 private IInstrumentDetailService instrumentDetailService;  
  
 @InjectMocks  
 private InstrumentDetailController instrumentDetailController;  
  
 @BeforeEach  
 public void setup() {  
 MockitoAnnotations.*openMocks*(this);  
 }  
  
 @Test  
 public void testCreateInstrumentDetail() {  
 InstrumentDetailDTO instrumentDetailDTO = new InstrumentDetailDTO();  
 instrumentDetailDTO.setId(1L);  
 instrumentDetailDTO.setDescription("Test description");  
  
 ResponseEntity<?> responseEntity = instrumentDetailController.createInstrumentDetail(instrumentDetailDTO);  
  
 *verify*(instrumentDetailService, *times*(1)).createInstrumentDetail(instrumentDetailDTO);  
 *assertEquals*(HttpStatus.*OK*, responseEntity.getStatusCode());  
 }  
  
 @Test  
 public void testGetInstrumentDetail() {  
 Long id = 1L;  
 InstrumentDetailDTO instrumentDetailDTO = new InstrumentDetailDTO();  
 instrumentDetailDTO.setId(id);  
 instrumentDetailDTO.setDescription("Test description");  
  
 *when*(instrumentDetailService.readInstrumentDetail(id)).thenReturn(instrumentDetailDTO);  
  
 InstrumentDetailDTO result = instrumentDetailController.getInstrumentDetail(id);  
  
 *verify*(instrumentDetailService, *times*(1)).readInstrumentDetail(id);  
 *assertEquals*(instrumentDetailDTO, result);  
 }  
  
 @Test  
 public void testModifyInstrumentDetail() {  
 InstrumentDetailDTO instrumentDetailDTO = new InstrumentDetailDTO();  
 instrumentDetailDTO.setId(1L);  
 instrumentDetailDTO.setDescription("Test description");  
  
 ResponseEntity<?> responseEntity = instrumentDetailController.modifyInstrumentDetail(instrumentDetailDTO);  
  
 *verify*(instrumentDetailService, *times*(1)).modifyInstrumentDetail(instrumentDetailDTO);  
 *assertEquals*(HttpStatus.*OK*, responseEntity.getStatusCode());  
 }  
  
 @Test  
 public void testRemoveInstrumentDetail() {  
 Long id = 1L;  
  
 ResponseEntity<?> responseEntity = instrumentDetailController.removeInstrumentDetail(id);  
  
 *verify*(instrumentDetailService, *times*(1)).removeInstrumentDetail(id);  
 *assertEquals*(HttpStatus.*OK*, responseEntity.getStatusCode());  
 }  
  
 @Test  
 public void testGetAllInstrumentDetails() {  
 Set<InstrumentDetailDTO> instrumentDetailDTOSet = new HashSet<>();  
 InstrumentDetailDTO instrumentDetailDTO1 = new InstrumentDetailDTO();  
 instrumentDetailDTO1.setId(1L);  
 instrumentDetailDTO1.setDescription("Test description 1");  
 InstrumentDetailDTO instrumentDetailDTO2 = new InstrumentDetailDTO();  
 instrumentDetailDTO2.setId(2L);  
 instrumentDetailDTO2.setDescription("Test description 2");  
 instrumentDetailDTOSet.add(instrumentDetailDTO1);  
 instrumentDetailDTOSet.add(instrumentDetailDTO2);  
  
 *when*(instrumentDetailService.getAll()).thenReturn(instrumentDetailDTOSet);  
  
 Set<InstrumentDetailDTO> result = (Set<InstrumentDetailDTO>) instrumentDetailController.getAllInstrumentDetails();  
  
 *verify*(instrumentDetailService, *times*(1)).getAll();  
 *assertEquals*(instrumentDetailDTOSet, result);  
 }  
}

