package com.nsw.backend.web.rest;  
  
import com.fasterxml.jackson.databind.ObjectMapper;  
import com.nsw.backend.service.BCTCRateService;  
import com.nsw.backend.service.dto.BctcRateDTO;  
import com.nsw.backend.web.rest.vm.ManagedUserVM;  
import org.hamcrest.Matchers;  
import org.junit.jupiter.api.Assertions;  
import org.junit.jupiter.api.BeforeEach;  
import org.junit.jupiter.api.Test;  
import org.junit.jupiter.api.extension.ExtendWith;  
import org.mockito.InjectMocks;  
import org.mockito.Mock;  
import org.mockito.Mockito;  
import org.mockito.junit.jupiter.MockitoExtension;  
import org.springframework.boot.test.json.JacksonTester;  
import org.springframework.data.domain.Page;  
import org.springframework.data.domain.PageImpl;  
import org.springframework.data.domain.PageRequest;  
import org.springframework.data.domain.Pageable;  
import org.springframework.http.HttpStatus;  
import org.springframework.http.MediaType;  
import org.springframework.mock.web.MockHttpServletResponse;  
import org.springframework.security.core.Authentication;  
import org.springframework.security.core.GrantedAuthority;  
import org.springframework.security.core.context.SecurityContextHolder;  
import org.springframework.test.web.servlet.MockMvc;  
import org.springframework.test.web.servlet.setup.MockMvcBuilders;  
  
import java.util.ArrayList;  
import java.util.Collection;  
  
import static org.hamcrest.MatcherAssert.*assertThat*;  
import static org.springframework.test.web.servlet.request.MockMvcRequestBuilders.*get*;  
import static org.springframework.test.web.servlet.request.MockMvcRequestBuilders.*post*;  
  
@ExtendWith(MockitoExtension.class)  
public class BctRateResourceTest {  
 private MockMvc mvc;  
  
 @InjectMocks  
 BctcRateResource bctcRateResource;  
  
 @Mock  
 BCTCRateService bctcRateService;  
  
 @BeforeEach  
 void setUp() {  
 SecurityContextHolder.*getContext*().setAuthentication(new Authentication() {  
 @Override  
 public Collection<? extends GrantedAuthority> getAuthorities() {  
 return null;  
 }  
  
 @Override  
 public Object getCredentials() {  
 return null;  
 }  
  
 @Override  
 public Object getDetails() {  
 return null;  
 }  
  
 @Override  
 public Object getPrincipal() {  
 return "test";  
 }  
  
 @Override  
 public boolean isAuthenticated() {  
 return false;  
 }  
  
 @Override  
 public void setAuthenticated(boolean b) throws IllegalArgumentException {  
  
 }  
  
 @Override  
 public String getName() {  
 return "test";  
 }  
 });  
  
 mvc = MockMvcBuilders.*standaloneSetup*(bctcRateResource).build();  
 JacksonTester.*initFields*(this, new ObjectMapper());  
 }  
  
 @Test  
 public void getAllBctcRate() throws Exception {  
 ManagedUserVM arg0 = new ManagedUserVM();  
 arg0.setPassword("password");  
 arg0.setLogin("test1");  
 arg0.setLangKey("vn");  
 MockHttpServletResponse response = mvc.perform(  
 *get*("/api/bctc-rate")  
 .accept(MediaType.*APPLICATION\_JSON*)  
 .content((new ObjectMapper()).writeValueAsString(arg0))  
 .contentType(MediaType.*APPLICATION\_JSON*))  
 .andReturn().getResponse();  
 *assertThat*(200, Matchers.*equalTo*(response.getStatus()));  
 }  
  
 @Test  
 public void checkExistConfig() throws Exception {  
 ManagedUserVM arg0 = new ManagedUserVM();  
 arg0.setPassword("password");  
 arg0.setLogin("test1");  
 arg0.setLangKey("vn");  
  
 MockHttpServletResponse response = mvc.perform(  
 *post*("/api/bctc-rate/exist")  
 .accept(MediaType.*APPLICATION\_JSON*)  
 .content((new ObjectMapper()).writeValueAsString(arg0))  
 .contentType(MediaType.*APPLICATION\_JSON*)  
 ).andReturn().getResponse();  
 *assertThat*(200, Matchers.*equalTo*(response.getStatus()));  
 }  
  
 @Test  
 public void getBctcRateByPages() throws Exception {  
 Pageable pageable = new PageRequest(0, 20);  
 BctcRateDTO dto = new BctcRateDTO();  
  
 Page<BctcRateDTO> page = new PageImpl<BctcRateDTO>(new ArrayList<>());  
 Mockito.*lenient*().when(bctcRateService.getAllAndSearch(Mockito.*any*(), Mockito.*any*()))  
 .thenReturn(page);  
  
 bctcRateResource.getAllBctcRates(pageable, dto);  
 *assertThat*(200, Matchers.*equalTo*(bctcRateResource.getAllBctcRates(pageable, dto).getStatusCode().value()));  
 }  
  
 @Test  
 public void getConfigById() throws Exception {  
 Mockito.*when*(bctcRateService.getById(Mockito.*any*())).thenReturn(new BctcRateDTO());  
 *assertThat*(200, Matchers.*equalTo*(bctcRateResource.getOneBctcRates(4l).getStatusCode().value()));;  
 }  
  
 @Test  
 public void deleteBctcConfigTest() throws Exception {  
// ManagedUserVM arg0 = new ManagedUserVM();  
// arg0.setPassword("password");  
// arg0.setLogin("test1");  
 Mockito.*when*(bctcRateService.delete(Mockito.*anyLong*())).thenReturn(4l);  
// MockHttpServletResponse response = mvc.perform(  
// get("/api" + "/bctc-rate/delete")  
// .accept(MediaType.APPLICATION\_JSON)  
// .content((new ObjectMapper())  
// .writeValueAsString(arg0))  
// .contentType(MediaType.APPLICATION\_JSON)  
// ).andReturn().getResponse();  
//  
// assertThat(response.getStatus(), Matchers.equalTo(HttpStatus.OK.value()));  
 Assertions.*assertNotEquals*(bctcRateResource.deleteById(1l), Matchers.*equalTo*(200));  
 }  
  
 @Test  
 public void addConfigTest() throws Exception {  
 Mockito.*when*(bctcRateService.save(Mockito.*any*())).thenReturn(new BctcRateDTO());  
 Assertions.*assertNotNull*(bctcRateResource.addConfig(new BctcRateDTO()));  
 }  
}