package com.gamingMatchMaker.gamingMatchMaker.controller;  
  
import com.gamingMatchMaker.gamingMatchMaker.controller.authorization.UserDetail;  
import com.gamingMatchMaker.gamingMatchMaker.controller.registration.CreateRegistrationRequest;  
import com.gamingMatchMaker.gamingMatchMaker.model.Location;  
import com.gamingMatchMaker.gamingMatchMaker.model.UserAuthentication;  
import com.gamingMatchMaker.gamingMatchMaker.model.UserRec;  
import com.gamingMatchMaker.gamingMatchMaker.service.LocationService.LocationService;  
import com.gamingMatchMaker.gamingMatchMaker.service.authService.UserAuthRecPair;  
import com.gamingMatchMaker.gamingMatchMaker.service.authService.UserAuthService;  
import com.gamingMatchMaker.gamingMatchMaker.service.registrationService.RegistrationService;  
import org.junit.Before;  
import org.junit.runner.RunWith;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.boot.test.autoconfigure.web.servlet.AutoConfigureMockMvc;  
import org.springframework.boot.test.context.SpringBootTest;  
import org.springframework.boot.test.mock.mockito.MockBean;  
import org.springframework.test.context.junit4.SpringRunner;  
import org.springframework.test.web.servlet.MockMvc;  
  
import java.io.BufferedReader;  
import java.io.FileNotFoundException;  
import java.io.InputStream;  
import java.io.InputStreamReader;  
import java.util.Optional;  
import java.util.UUID;  
import java.util.stream.Collectors;  
  
import static org.mockito.ArgumentMatchers.\*;  
import static org.mockito.ArgumentMatchers.*anyInt*;  
import static org.mockito.ArgumentMatchers.*eq*;  
import static org.mockito.Mockito.*spy*;  
import static org.mockito.Mockito.*when*;  
  
@RunWith(SpringRunner.class)  
@SpringBootTest  
@AutoConfigureMockMvc  
public abstract class ControllerTest {  
  
 protected static final String *TEST\_USERNAME* = "user1";  
 protected static final String *TEST\_PASSWORD* = "pass1";  
  
 @Autowired  
 protected MockMvc mockMvc;  
  
 @MockBean  
 private UserAuthService userAuthService;  
  
 @MockBean  
 private RegistrationService registrationService;  
  
 @MockBean  
 private LocationService locationService;  
  
 @Before  
 public void setupUserAuthService() {  
 // build the mock server responses  
 Location mockLocation = new Location("30047", "liburn","GA",  
 50.30f,25.5f, "test is a test");  
  
 UserRec mockUserRec = new UserRec("test@aaa.com","test",  
 "test","test",1,true,2, mockLocation);  
  
 UserAuthentication mockAuth = new UserAuthentication(1,  
 UUID.*fromString*("ee97d265-4828-4aa5-8b7c-1ab41b03b405"), 1);  
  
 UserAuthRecPair mockAuthRecPair = new UserAuthRecPair(mockAuth, mockUserRec);  
  
 *when*(userAuthService.authByEmailPassword(*TEST\_USERNAME*, *TEST\_PASSWORD*))  
 .thenReturn(mockAuthRecPair);  
 }  
  
 @Before  
 public void setupRegistrationService() {  
 Location mockLocation = new Location("99004","testCiy","NY",  
 20.05f,15.50f,"Jons Test Location");  
 mockLocation.setId(0);  
 UserDetail mockUserDetail = new UserDetail(  
 0,"testJon@test.com","Jon","Doe",  
 37,true,1,mockLocation);  
 UserRec mockUserRec = new UserRec(mockUserDetail);  
 CreateRegistrationRequest request =  
 new CreateRegistrationRequest(mockUserDetail, "password1");  
  
 mockUserDetail.getId();  
  
 Location mockLocationRec = new Location(mockLocation);  
 mockLocationRec.setId(42);  
 UserRec response = new UserRec(mockUserDetail);  
 response.setId(43);  
 response.setLocation(mockLocationRec);  
  
 *when*(this.locationService.GetLocation(*eq*(mockLocation.getZip()))).thenReturn(mockLocation);  
  
 *when*(  
 registrationService.createRegistration(  
 *eq*(mockUserRec), *eq*("password1")  
 )  
 ).thenReturn(Optional.*of*(response));  
 }  
  
 protected String readFileFromResources(Class c, String fileName) throws Exception {  
 String result;  
  
 if(c == null) {  
 throw new IllegalArgumentException("Class cannot be null");  
 }  
  
 String pathName = c.getPackage().getName().replace('.', '/') + "/" + fileName;  
  
 InputStream inputStream =  
 this.getClass().getClassLoader().getResourceAsStream(pathName);  
  
 if(inputStream == null) {  
 throw new FileNotFoundException("No file found in classpath for " + pathName);  
 }  
  
 result = new BufferedReader(new InputStreamReader(inputStream))  
 .lines().collect(Collectors.*joining*("\n"));  
  
 return result;  
 }  
}

package com.gamingMatchMaker.gamingMatchMaker.controller.registration;  
  
 import com.gamingMatchMaker.gamingMatchMaker.service.registrationService.RegistrationService;  
 import org.junit.Before;  
 import org.junit.Test;  
 import org.junit.runner.RunWith;  
 import org.springframework.beans.factory.annotation.Autowired;  
 import org.springframework.boot.test.autoconfigure.web.servlet.AutoConfigureMockMvc;  
 import org.springframework.boot.test.context.SpringBootTest;  
 import org.springframework.boot.test.mock.mockito.MockBean;  
 import org.springframework.http.MediaType;  
 import org.springframework.test.context.junit4.SpringRunner;  
 import org.springframework.test.web.servlet.MockMvc;  
 import org.springframework.test.web.servlet.RequestBuilder;  
 import org.springframework.test.web.servlet.request.MockMvcRequestBuilders;  
  
 import static org.springframework.test.web.servlet.result.MockMvcResultMatchers.*content*;  
 import static org.springframework.test.web.servlet.result.MockMvcResultMatchers.*status*;  
  
@RunWith(SpringRunner.class)  
@SpringBootTest  
@AutoConfigureMockMvc  
public class RegistrationControllerTest extends com.gamingMatchMaker.gamingMatchMaker.controller.ControllerTest {  
  
 @Test  
 public void testRegisterUserSuccess() throws Exception {  
  
 // usually for testing read mock JSON's from a file  
  
 String mockRequestJSON = readFileFromResources(  
 RegistrationController.class,  
 "registerUserSuccessRequest.json"  
 );  
 String mockResponseJSON = readFileFromResources(  
 RegistrationController.class,  
 "registerUserSuccessResponse.json"  
 );  
  
 RequestBuilder requestBuilder = MockMvcRequestBuilders.*post*("/api/register")  
 .accept(MediaType.*APPLICATION\_JSON*)  
 .contentType(MediaType.*APPLICATION\_JSON*)  
 .content(mockRequestJSON);  
  
 mockMvc.perform(requestBuilder)  
 .andExpect(*status*().isOk())  
 .andExpect(*content*().json(mockResponseJSON));  
 }  
  
  
}