package com.gamingMatchMaker.gamingMatchMaker.service;  
  
  
import com.gamingMatchMaker.gamingMatchMaker.dao.LocationRepository;  
import com.gamingMatchMaker.gamingMatchMaker.dao.UserAuthenticationRepository;  
import com.gamingMatchMaker.gamingMatchMaker.dao.UserRepository;  
import com.gamingMatchMaker.gamingMatchMaker.model.Location;  
import com.gamingMatchMaker.gamingMatchMaker.model.UserAuthentication;  
import com.gamingMatchMaker.gamingMatchMaker.model.UserRec;  
import com.gamingMatchMaker.gamingMatchMaker.model.UserType;  
import com.gamingMatchMaker.gamingMatchMaker.service.authService.UserException;  
import org.hibernate.id.UUIDGenerator;  
import org.junit.Before;  
import org.junit.runner.RunWith;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.boot.test.context.SpringBootTest;  
import org.springframework.boot.test.mock.mockito.MockBean;  
import org.springframework.security.crypto.password.PasswordEncoder;  
import org.springframework.test.context.TestPropertySource;  
import org.springframework.test.context.junit4.SpringRunner;  
  
import java.util.Arrays;  
import java.util.List;  
import java.util.Optional;  
import java.util.UUID;  
  
import static org.mockito.ArgumentMatchers.*any*;  
import static org.mockito.Mockito.*when*;  
  
@RunWith(SpringRunner.class)  
@TestPropertySource(locations="classpath:test.properties")  
@SpringBootTest  
public abstract class ServiceTest {  
 @MockBean  
 private UserRepository userDao;  
  
 @MockBean  
 private UserAuthenticationRepository authDao;  
  
 @MockBean  
 private LocationRepository locDao;  
  
 @Autowired  
 private PasswordEncoder passwordEncoder;  
  
 // Known Id to represent the successful creation of a UserRec or UserAuthentication  
 protected static final int *ID\_SUCCESS* = 101;  
 protected static final UUID *UUID\_SUCCESS* = UUID.*randomUUID*();  
  
 // Known Locations  
 public static final Location *LOC\_1* = new Location(1, "90210", "witerfell", "the North",  
 3f, 1f, "Winterfel Castle in the north");  
 public static final Location *LOC\_2* = new Location(2, "30221", "galaxy", "death star",  
 1f, 5f, "Darth Vader house");  
 public static final Location *LOC\_3* = new Location(3, "30047", "galaxy", "death star",  
 1f, 5f, "Darth Vader house");  
  
  
 // Known set of users for testing  
 public static final UserRec *USER\_1* = new UserRec(  
 "user1@testDomain.tst",  
 "Ned", "stark",  
 "Winter\*I3\_CominG!",  
 37, true, UserType.*admin*.getValue(),  
 *LOC\_1*);  
 public static final UserRec *USER\_2* = new UserRec(  
 "user2@testDomain.tst",  
 "Han", "solo",  
 "password",  
 37, true, UserType.*player*.getValue(),  
 *LOC\_2*);  
  
 @Before  
 public void setupLocationRepositoryKnownUsersForFind() {  
 List<Location> locationList = Arrays.*asList*(*LOC\_1*, *LOC\_2*, *LOC\_3*);  
  
 for(Location loc: locationList) {  
 *when*(locDao.findByZip(loc.getZip()))  
 .thenReturn(Optional.*of*(loc));  
 }  
 }  
  
  
 @Before  
 public void setupUserRepositoryCreateScenarios() {  
 // deal with the error conditions first  
  
 // mock duplicate email creation fail  
 List<UserRec> knownUsers = Arrays.*asList*(  
 *USER\_1*, *USER\_2* );  
 for (UserRec user : knownUsers) {  
 *when*(userDao.save(user))  
 .thenThrow(new UserException("Duplicate email"));  
 }  
  
 // catchall for success  
 *when*(userDao.save(*any*(UserRec.class)))  
 .thenAnswer(  
 // use a lambda function to modify the return value (based on the input value)  
 invocation -> {  
 // first get the input UserRec  
 UserRec inputUserRec = invocation.getArgument(0);  
 // make a copy  
 UserRec resultUserRec = new UserRec(inputUserRec);  
  
 // modify the copy to simulate the user dao return value  
  
 // give it a known ID  
 resultUserRec.setId(*ID\_SUCCESS*);  
  
 // also set any default values here  
  
 // return the copy  
 return resultUserRec;  
 }  
 );  
 }  
  
 @Before  
 public void setupUserRepositoryKnownUsersForFind() {  
 List<UserRec> userList = Arrays.*asList*(*USER\_1*, *USER\_2*);  
  
 for(UserRec user: userList) {  
 UserRec tmp = new UserRec(user);  
  
 tmp.setPassword(passwordEncoder.encode(user.getPassword()));  
 *when*(userDao.findByEmail(user.getEmail()))  
 .thenReturn(Optional.*of*(tmp));  
 }  
 }  
  
 @Before  
 public void setupUserAuthenticationRepository() {  
 *when*(authDao.save(*any*(UserAuthentication.class)))  
 .thenAnswer(  
 invocation -> {  
 UserAuthentication userAuth = invocation.getArgument(0);  
  
 UserAuthentication resultUserAuth = new UserAuthentication(userAuth);  
  
 resultUserAuth.setAccessToken(*UUID\_SUCCESS*);  
  
 return resultUserAuth;  
 }  
 );  
 }  
}

package com.gamingMatchMaker.gamingMatchMaker.service.authService;  
  
import com.gamingMatchMaker.gamingMatchMaker.model.UserAuthentication;  
import com.gamingMatchMaker.gamingMatchMaker.service.ServiceTest;  
import org.junit.Test;  
import org.springframework.beans.factory.annotation.Autowired;  
  
import static org.junit.Assert.\*;  
  
public class UserAuthServiceImplTest extends ServiceTest {  
 @Autowired  
 private UserAuthService authService;  
  
 @Test  
 public void authByEmailPassword() {  
 UserAuthRecPair result = this.authService.authByEmailPassword(*USER\_1*.getEmail(), *USER\_1*.getPassword());  
  
 *assertNotNull*(result);  
 *assertNotNull*(result.getUserRec());  
 *assertNotNull*(result.getAuth());  
  
 // expected then actual  
 *assertEquals*(*USER\_1*.getEmail(), result.getUserRec().getEmail());  
 // assertEquals(USER\_1.getId(), result.getUserRec().getId());  
 *assertEquals*(*USER\_1*.getUser\_type(), result.getUserRec().getUser\_type());  
  
 *assertEquals*(*UUID\_SUCCESS*, result.getAuth().getAccessToken());  
 //assertEquals(USER\_1.getId(), result.getAuth().getUserId());  
 }  
  
  
}

package com.gamingMatchMaker.gamingMatchMaker.service.registrationService;  
  
import com.gamingMatchMaker.gamingMatchMaker.model.Location;  
import com.gamingMatchMaker.gamingMatchMaker.model.UserRec;  
import com.gamingMatchMaker.gamingMatchMaker.service.LocationService.LocationService;  
import com.gamingMatchMaker.gamingMatchMaker.service.ServiceTest;  
import org.junit.Test;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.boot.test.mock.mockito.MockBean;  
  
import java.util.Optional;  
  
import static junit.framework.TestCase.*assertTrue*;  
import static org.junit.Assert.*assertEquals*;  
import static org.mockito.Mockito.*when*;  
  
public class RegistrationServiceImplTest extends ServiceTest {  
  
 @Autowired  
 private RegistrationService registrationService;  
  
 @MockBean  
 private LocationService locationService;  
  
 @Test  
 public void testCreateRegistrationSuccess() {  
  
 // make sure that location search returns the correct location  
 *when*(locationService.GetLocation(*LOC\_1*.getZip())).thenReturn(*LOC\_1*);  
  
 // simulate a call to the service to create a user registration  
  
 // create a new newUserRecDetails, and s password  
 UserRec newUserRecDetails = new UserRec(  
 "test1@test.com", "test1", "test",  
 "password", 22, true, 1, *LOC\_1*);  
  
 String password = "test";  
  
 // call the service method  
 Optional<UserRec> result = this.registrationService.createRegistration(newUserRecDetails, password);  
  
 // assert to make sure user reg was created  
 *assertTrue*(result.isPresent());  
 if (result.isPresent()) {  
 *assertEquals*(ServiceTest.*ID\_SUCCESS*, result.get().getId());  
 }  
 }  
  
}