Лабораторная работа №6

по дисциплине «*Модульное тестирование*»

«ИНТЕГРАЦИОННОЕ ТЕСТИРОВАНИЕ»

**Цель работы:** закрепление практических навыков по проведению интеграционного тестирования.

**Задание:** провести автоматизацию тестов для проверки преобразования данных для рассмотренных ранее модулей согласно описанной последовательности:

1. Разработать интеграционные тесты для методов, используемых в предыдущих лабораторных работах. Предусмотреть проверки вызовов определенных функций внутри тестируемого метода**.**

2. Запустить разработанные тесты. Занести результаты пункта в отчет в виде копий экранов с не пройденными тестами и листингом кода.

3. Произвести разработку кода, не прибегая к изменению тестов. Запустить тесты. Занести результаты пункта в отчет в виде копий экранов с результатами тестов и листингом разработанного кода.

4. Произвести рефакторинг кода и тестов. В отчет занести изменения по коду и комментарии к ним.

5. Сделать выводы по работе.

Программный модуль admin-a

и незарегистрированного user-а сайта «*BuyingHouses*»

**Автор:** студент гр.10701118 Дубоделов А.В.

**Проверил:** доцент к.т.н. Попова Ю.Б.

Содержание

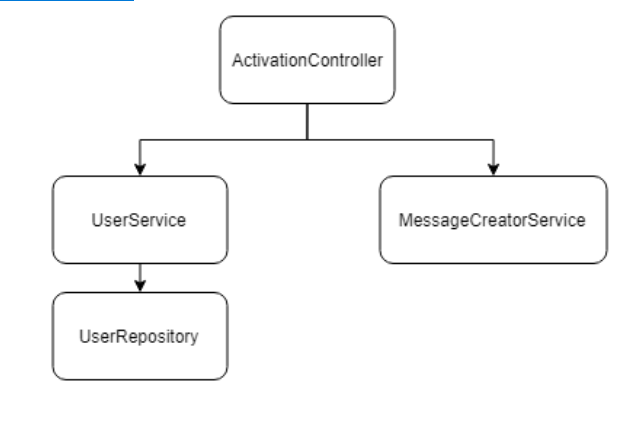
[Структура взаимодействия классов. 3](#_Toc57406039)

[Написание интегрированных тестов. 3](#_Toc57406040)

[Результаты проведения тестов. 11](#_Toc57406041)

[Вывод 12](#_Toc57406042)

# **Структура взаимодействия классов.**

Для выполнения лабораторной работы за основу была принята следующая цепочка классов:  
  


Для разработки тестов было принято выбрать восходящий подход.

# **Написание интегрированных тестов.**

Разработка тестов началась с левой части цепочки, так как входные данные берутся из БД, данные из которой получаются посредством методов UserRepository.

Были разработаны тесты связывающиеся с существующей(реальной БД проекта, после каждого отработанного теста происходит откат БД)

|  |
| --- |
| @RunWith(SpringRunner.**class**) @SpringBootTest @Transactional **public class** UserRepositoryTest {   @Autowired  **private** UserRepository **userRepository**;   **private static** String *activationCode*;  **private static** String *nonActivationCode*;  **private static** User *user*;   @BeforeClass  **public static void** init(){  *activationCode* = **"37212680-2e8e-45c4-82d9-f213b587130d"**;;  *nonActivationCode* = **"non"**;  *user* = **new** User();   *user*.setPassword(**"testName"**);  *user*.setUserName(**"testName"**);  *user*.setEmail(**"something@some.some"**);  *user*.setActivationCode(*activationCode*);  *user*.setActive(**false**);  *user*.setRoles(Collections.*singleton*(Role.***USER***));  }   @Test  **public void** findByActivationCode\_activationCode\_user() {  User currentUser = **userRepository**.findByActivationCode(*activationCode*);   Assert.*assertNotNull*(currentUser);  }   @Test  **public void** findByActivationCode\_nonActivationCode\_null() {  User currentUser = **userRepository**.findByActivationCode(*nonActivationCode*);   Assert.*assertNull*(currentUser);  }   @Test  **public void** save\_user\_userWithId() {  User currentUser = **userRepository**.save(*user*);   Assert.*assertNotNull*(currentUser.getId());  } } |

После написанных тестов, необходимо было написать интегрированные тесты связав методы репозитория с методами UserService. Изначально были написаны тесты UserService с mock-заглушками для проверки факта вызова методов UserRepository внутри методов UserService, а также для проверки работы функционала методов UserService.

|  |
| --- |
| @RunWith(SpringRunner.**class**) @SpringBootTest **public class** MockUserServiceTest {   @Mock  **private** UserRepository **userRepository**;   @InjectMocks  **private** UserService **userService**;   **private** String **activationCode**;  **private** User **user**;   @Before  **public void** init(){  **activationCode** = **"1234"**;  **user** = **new** User();  }   @Test  **public void** activateUser\_activationCode\_true(){  *when*(**userRepository**.findByActivationCode(**activationCode**)).thenReturn(**user**);  *when*(**userRepository**.save(**user**)).thenReturn(**user**);   **boolean** result = **userService**.activateUser(**activationCode**);   Assert.*assertTrue*(result);  }   @Test  **public void** activateUser\_activationCode\_false(){  *when*(**userRepository**.findByActivationCode(**activationCode**)).thenReturn(**null**);  *when*(**userRepository**.save(**user**)).thenReturn(**null**);   **boolean** result = **userService**.activateUser(**activationCode**);   Assert.*assertFalse*(result);  }   @Test  **public void** save\_user\_user(){  **user**.setId(1L);  *when*(**userRepository**.save(**user**)).thenReturn(**user**);   User result = **userService**.save(**user**);   Assert.*assertNotNull*(result);  }   @Test  **public void** activateUser\_verify\_statistic(){  *when*(**userRepository**.findByActivationCode(**activationCode**)).thenReturn(**user**);  *when*(**userRepository**.save(**user**)).thenReturn(**user**);   **userService**.activateUser(**activationCode**);   Mockito.*verify*(**userRepository**).findByActivationCode(**activationCode**);  Mockito.*verify*(**userRepository**).save(**user**);  } } |

После чего были написаны интеграционные тесты между этими классами, без использования заглушек.

|  |
| --- |
| @RunWith(SpringRunner.**class**) @SpringBootTest @Transactional **public class** AutowiredUserServiceTest {   @Autowired  **private** UserService **userService**;   **private static** String *activationCode*;  **private static** String *nonActivationCode*;  **private static** User *user*;   @BeforeClass  **public static void** init(){  *activationCode* = **"37212680-2e8e-45c4-82d9-f213b587130d"**;  *nonActivationCode* = **"none"**;  *user* = **new** User();   *user*.setPassword(**"testName"**);  *user*.setUserName(**"testName"**);  *user*.setEmail(**"something@some.some"**);  *user*.setActivationCode(*activationCode*);  *user*.setActive(**false**);  *user*.setRoles(Collections.*singleton*(Role.***USER***));  }   @Test  **public void** activateUser\_activationCode\_true(){  **boolean** result = **userService**.activateUser(*activationCode*);   Assert.*assertTrue*(result);  }   @Test  **public void** activateUser\_activationCode\_false(){  **boolean** result = **userService**.activateUser(*nonActivationCode*);   Assert.*assertFalse*(result);  }   @Test  **public void** save\_user\_user(){  User result = **userService**.save(*user*);   Assert.*assertNotNull*(result);  }  } |

Далее были реализованы (взят из 3-ей лабораторной работы) тесты класса MessageCreatorService.

|  |
| --- |
| @RunWith(SpringRunner.**class**) @SpringBootTest **public class** MessageCreatorServiceTest {   @Autowired  **private** MessageCreatorService **creatorService**;   @Test  **public void** createlMessage\_getEmptyPasswordMessage\_PleaseFillFieldRepeatedPassword() {   String result = **creatorService**.createMessage(Messages.***EMPTY\_PASSWORD\_MESSAGE***);  String expected = **"Please fill field repeated password"**;   Assert.*assertEquals*(expected, result);  }   @Test  **public void** createMessage\_getDifferentPasswordMessage\_PasswordsAreDifferent() {   String result = **creatorService**.createMessage(Messages.***DIFFERENT\_PASSWORD\_MESSAGE***);  String expected = **"Passwords are different"**;   Assert.*assertEquals*(expected, result);  }   @Test  **public void** createMessage\_getUserExistMessage\_UserWithTheSameEmailLoginAlreadyExist() {   String result = **creatorService**.createMessage(Messages.***USER\_EXIST\_MESSAGE***);  String expected = **"User with the same email/login already exist"**;   Assert.*assertEquals*(expected, result);  }   @Test  **public void** createHessage\_getSuccessfullyActivatedMessage\_UserSuccessfullyActivated() {   String result = **creatorService**.createMessage(Messages.***SUCCESSFULLY\_ACTIVATED\_MESSAGE***);  String expected = **"User successfully activated"**;   Assert.*assertEquals*(expected, result);  }   @Test  **public void** createMessage\_getActivationCodeNotFoundMessage\_ActivationCodeIsNotFound() {   String result = **creatorService**.createMessage(Messages.***ACTIVATION\_CODE\_NOT\_FOUND\_MESSAGE***);  String expected = **"Activation code is not found"**;   Assert.*assertEquals*(expected, result);  }   @Test  **public void** createMessage\_ACCOMMODATION\_EXIST\_MESSAGE\_AccommodationWithTheSameNameAlreadyExists() {   MessageCreatorService creatorService = **new** MessageCreatorService();    String result = creatorService.createMessage(Messages.***ACCOMMODATION\_EXIST\_MESSAGE***);  String expected = **"Accommodation with the same name already exists"**;    Assert.*assertEquals*(expected, result);  }   @Test  **public void** createMessage\_getDefaultMessage\_BadRequest() {   String result = **creatorService**.createMessage(Messages.***DEFAULT***);  String expected = **"Bad request"**;   Assert.*assertEquals*(expected, result);  } } |

После чего были реализованы тесты методов ActivationController.

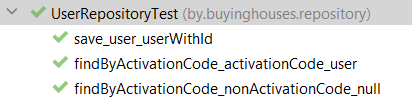
Изначально были реализованы тесты с использованием mock элементов, для проверки факта вызова методов UserService и MessageCreatorService в контроллере.

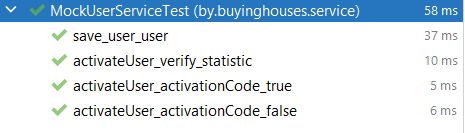
|  |
| --- |
| @RunWith(SpringRunner.**class**) @SpringBootTest @Transactional @AutoConfigureMockMvc **public class** ActivationControllerTest {   @Mock  **private** UserRepository **userRepository**;   @Mock  **private** UserService **userService**;   @Mock  **private** MessageCreatorService **messageCreatorService**;   @InjectMocks  **private** ActivationController **activationController**;   **private static final** String ***ACTIVATION\_CODE*** = **"37212680-2e8e-45c4-82d9-f213b587130d"**;  **private static final** String ***NON\_ACTIVATION\_CODE*** = **"12"**;  **private static final** User ***USER*** = **new** User();  @Test  **public void** activate\_successfulVerify\_statistic(){  *when*(**userRepository**.findByActivationCode(***ACTIVATION\_CODE***)).thenReturn(***USER***);  *when*(**userRepository**.save(***USER***)).thenReturn(***USER***);  *when*(**userService**.findByActivationCode(***ACTIVATION\_CODE***)).thenReturn(***USER***);  *when*(**userService**.save(***USER***)).thenReturn(***USER***);  *when*(**userService**.activateUser(***ACTIVATION\_CODE***)).thenReturn(**true**);  *when*(**messageCreatorService**.createMessage(Messages.***SUCCESSFULLY\_ACTIVATED\_MESSAGE***)).thenReturn(**"User successfully activated"**);   **activationController**.activate(**new** ConcurrentModel(), ***ACTIVATION\_CODE***);   Mockito.*verify*(**userService**).activateUser(***ACTIVATION\_CODE***);  Mockito.*verify*(**messageCreatorService**).createMessage(Messages.***SUCCESSFULLY\_ACTIVATED\_MESSAGE***);  }   @Test  **public void** activate\_unsuccessfulVerify\_statistic(){  *when*(**userRepository**.findByActivationCode(***NON\_ACTIVATION\_CODE***)).thenReturn(***USER***);  *when*(**userRepository**.save(***USER***)).thenReturn(***USER***);  *when*(**userService**.findByActivationCode(***NON\_ACTIVATION\_CODE***)).thenReturn(***USER***);  *when*(**userService**.save(***USER***)).thenReturn(***USER***);  *when*(**userService**.activateUser(***NON\_ACTIVATION\_CODE***)).thenReturn(**false**);  *when*(**messageCreatorService**.createMessage(Messages.***ACTIVATION\_CODE\_NOT\_FOUND\_MESSAGE***)).thenReturn(**"Activation code is not found"**);   **activationController**.activate(**new** ConcurrentModel(), ***NON\_ACTIVATION\_CODE***);   Mockito.*verify*(**userService**).activateUser(***NON\_ACTIVATION\_CODE***);  Mockito.*verify*(**messageCreatorService**).createMessage(Messages.***ACTIVATION\_CODE\_NOT\_FOUND\_MESSAGE***);  } |

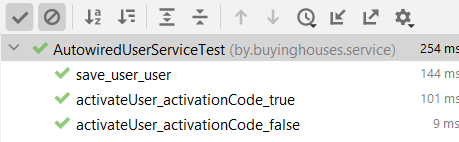
Для интеграционного тестирования RESTful контроллеров в Java Spring применяется объект MockMvc, он позволяет тестировать данные классы без запуска сервера. То есть при выполнении тестов сетевое соединение не создается.

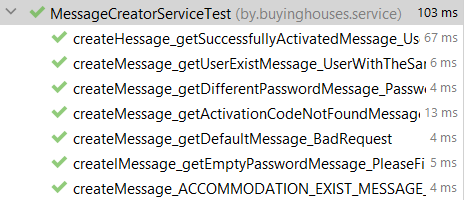
|  |
| --- |
| @RunWith(SpringRunner.**class**) @SpringBootTest @Transactional @AutoConfigureMockMvc **public class** ActivationControllerTest {   @Autowired  **private** MockMvc **mockMvc**;  **private static final** String ***ACTIVATION\_CODE*** = **"37212680-2e8e-45c4-82d9- f213b587130d"**;  **private static final** String ***NON\_ACTIVATION\_CODE*** = **"12"**;  **private static final** User ***USER*** = **new** User();   @Test  **public void** activate\_id\_statusOkNotActivate() **throws** Exception {  **this**.**mockMvc**.perform(*get*(**"/activate/"** + ***NON\_ACTIVATION\_CODE***))  .andDo(*print*())  .andExpect(*status*().isOk())  .andExpect(*content*().string(*containsString*(**"Activation code is not found"**)));  }   @Test  **public void** activate\_id\_statusOkActivate() **throws** Exception {  **this**.**mockMvc**.perform(*get*(**"/activate/"** + ***ACTIVATION\_CODE***))  .andDo(*print*())  .andExpect(*status*().isOk())  .andExpect(*content*().string(*containsString*(**"User successfully activated"**)));  }  @Test  **public void** activate\_id\_exception() **throws** AssertionError{  *assertThrows*(AssertionError.**class**, () -> {  **this**.**mockMvc**.perform(*get*(**"/activate"**))  .andDo(*print*())  .andExpect(*status*().isOk())  .andExpect(*content*().string(*containsString*(**"Other"**))); });  } |

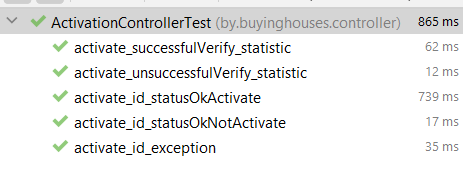
# **Результаты проведения тестов.**











В результате выполнения, все тесты из данной цепочки дали положительный результат. В следствии чего, рефакторинг кода не производился.

# **Вывод:** в ходе лабораторной работы была изучена восходящая концепция написания интеграционных тестов. Была выбрана начальная цепочка классов, к каждому из которых были написаны тесты с заглушками и без них. Далее были протестированы связки классов сначала UserRepository + UserService, далее UserRepository+UserService+ActivationController. При тестировании с заглушками были проверены факты вызовов методов, а также изолированная работа отдельных компонентов, в интеграционных тестах была протестирована правильная реализация взаимодействия различных методов одной цепочки.