МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«БГТУ им. В.Г.ШУХОВА»

Кафедра программного обеспечения вычислительной техники и автоматизированных систем

КУРСОВАЯ РАБОТА

по дисциплине:

«Тестирование программных систем»

тема: интеграционное тестирование Веб-приложения

Автор работы \_\_\_\_\_\_\_\_\_\_\_Донцов Александр Алексеевич, ПВ-41

(подпись)

Руководитель проекта\_\_\_\_\_\_\_\_\_\_\_\_\_ Поляков Владимир Константинович

(подпись)

Оценка \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Белгород

2020 г.

**Оглавление**

[Введение 3](#_Toc60864761)

[Постановка задачи 3](#_Toc60864762)

[Теоретические сведения 3](#_Toc60864763)

[Глава 1. Интеграционное тестирование 3](#_Toc60864764)

[Глава 2. Selenium WebDriver 4](#_Toc60864765)

[Глава 3. Процесс создания тестов 4](#_Toc60864766)

[**Заключение** 4](#_Toc60864767)

[Список использованной литературы 5](#_Toc60864768)

[Приложение 5](#_Toc60864769)

[**tests.js** 5](#_Toc60864770)

[**Скриншоты** 8](#_Toc60864771)

# Введение

С развитием веб-технологий появилась необходимость в интеграционном тестировании веб-приложений. Одним из средств являются тестирование апи контроллеров, а также тестирование внутренних сервисов проекта.

# Постановка задачи

Цель – интеграционное тестирование веб-приложения.

# Теоретические сведения

Интеграционное тестирование веб-приложения включает в себя эмуляцию пользовательских историй, то есть симуляция поведения пользователя с помощью программных средств

# Глава 1. Интеграционное тестирование

**Интеграционное тестирование -** одна из фаз тестирования программного обеспечения, при которой отдельные программные модули объединяются и тестируются в группе. Обычно интеграционное тестирование проводится после модульного тестирования и предшествует системному тестированию.

Интеграционное тестирование в качестве входных данных использует модули, над которыми было проведено модульное тестирование, группирует их в более крупные множества, выполняет тесты, определённые в плане тестирования для этих множеств, и представляет их в качестве выходных данных и входных для последующего системного тестирования.

Целью интеграционного тестирования является проверка соответствия проектируемых единиц функциональным, приёмным и требованиям надежности. Тестирование этих проектируемых единиц — объединения, множества или группы модулей — выполняется через их интерфейс, с использованием тестирования «чёрного ящика».

# Глава 2. Mockito

Mockito – фреймворк, помогающий создавать интеграционные тесты, он позволяет создавать имитацию запросов клиента на сервер и проверить данные, которые он возвращает.

# Глава 3. Процесс создания тестов

Для выполнения работы был выбран проект по вебу из предыдущих лабораторных. Он содержит необходимые списки сущностей, сами сущности и достаточное количество кнопок для проверки и изучения интеграционного тестирование веб-приложения.

## **Заключение**

В ходе выполнения курсовой работы были получены навыки в интеграционном тестировании.

# Список использованной литературы

1. **Wikipedia. Интеграционное тестирование** [Электронный ресурс] – Режим доступа: <https://ru.wikipedia.org/wiki/Интеграционное_тестирование>;
2. **Mockito и как его готовить**[Электронный ресурс] https://habr.com/ru/post/444982/

# Приложение

package back.controllers;  
  
  
import lombok.SneakyThrows;  
import org.junit.Test;  
import org.springframework.http.MediaType;  
  
import static org.springframework.test.web.servlet.request.MockMvcRequestBuilders.\*;  
import static org.springframework.test.web.servlet.result.MockMvcResultMatchers.*status*;  
  
public class BookControllerTest extends ApiBaseTest {  
  
 @SneakyThrows  
 @Test  
 public void addBook() {  
 mockMvc.perform(*post*("/api/v1/book").content(getBookDto(null)).contentType(MediaType.*APPLICATION\_JSON*)).andExpect(*status*().isOk());  
 }  
  
 @SneakyThrows  
 @Test  
 public void addChapterBook() {  
 mockMvc.perform(*post*("/api/v1/book/chapter").contentType(MediaType.*APPLICATION\_JSON*).content(getChapterBookDto())).andExpect(*status*().isOk());  
 }  
  
 @SneakyThrows  
 @Test  
 public void getBookChapter() {  
 mockMvc.perform(*get*("/api/v1/book/chapter/1/0")).andExpect(*status*().isOk());  
 }  
  
 @SneakyThrows  
 @Test  
 public void deleteBook() {  
 mockMvc.perform(*delete*("/api/v1/book/3")).andExpect(*status*().isOk());  
 }  
  
 @SneakyThrows  
 @Test  
 public void deleteChapterBook() {  
 mockMvc.perform(*delete*("/api/v1/book/chapter").param("bookId", "2").param("chapterId", "0")).andExpect(*status*().isOk());  
 }  
  
 @SneakyThrows  
 @Test  
 public void updateBook() {  
 mockMvc.perform(*put*("/api/v1/book").content(getBookDto(2L)).contentType(MediaType.*APPLICATION\_JSON*)).andExpect(*status*().isOk());  
 }  
  
 @SneakyThrows  
 @Test  
 public void getAllBook() {  
 mockMvc.perform(*get*("/api/v1/book/all")).andExpect(*status*().isOk());  
 }  
  
 @SneakyThrows  
 @Test  
 public void download() {  
 mockMvc.perform(*get*("/api/v1/book/logo/1")).andExpect(*status*().isOk());  
 }  
}

package back.service.impl;  
  
import back.dto.book.BookDto;  
import back.dto.book.ChapterBookDto;  
import back.dto.book.ShortBookDto;  
import back.service.BookService;  
import lombok.SneakyThrows;  
import org.junit.jupiter.api.Assertions;  
import org.junit.jupiter.api.Test;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.core.io.ClassPathResource;  
import org.springframework.http.MediaType;  
import org.springframework.mock.web.MockMultipartFile;  
import org.springframework.web.multipart.MultipartFile;  
  
import java.util.List;  
  
class BookServiceImplTest extends BaseTest {  
  
 @Autowired  
 private BookService bookService;  
  
 @Test  
 @SneakyThrows  
 void createBook() {  
 BookDto book = new BookDto();  
 book.setDescription("test");  
 book.setAuthor("test");  
 book.setName("test");  
  
 MultipartFile logo = new MockMultipartFile(  
 "test", "test.jpg",  
 MediaType.*IMAGE\_PNG\_VALUE*,  
 new ClassPathResource("unknown-book.jpg", this.getClass().getClassLoader()).getInputStream());  
  
 Long id = bookService.createBook(book, logo);  
 Assertions.*assertNotNull*(id);  
 }  
  
 @Test  
 void addPageBook() {  
 ChapterBookDto book = new ChapterBookDto();  
 book.setBookId(3L);  
 book.setNumChapter(1L);  
 book.setText("test");  
  
 Long id = bookService.addPageBook(book);  
 Assertions.*assertNotNull*(id);  
 }  
  
 @Test  
 void getAllBook() {  
 List<ShortBookDto> list = bookService.getAllBook();  
 Assertions.*assertNotNull*(list);  
 }  
  
 @Test  
 void getBookChapter() {  
 ChapterBookDto dto = bookService.getBookChapter(3L, 0L);  
 Assertions.*assertNotNull*(dto);  
 }  
  
 @Test  
 void deleteBook() {  
 bookService.deleteBook(4L);  
 }  
  
 @Test  
 void deleteChapterBook() {  
 bookService.deleteChapterBook(3L, 0L);  
 }  
  
 @Test  
 void updateBook() {  
 BookDto dto = new BookDto();  
 dto.setId(1L);  
 dto.setName("test");  
 dto.setDescription("test");  
 Long id = bookService.updateBook(dto);  
 Assertions.*assertNotNull*(id);  
 }  
  
 @Test  
 void updatePageBook() {  
 ChapterBookDto dto = new ChapterBookDto();  
 dto.setBookId(1L);  
 dto.setText("test");  
 dto.setNumChapter(1L);  
 Long id = bookService.updatePageBook(dto);  
 Assertions.*assertNotNull*(id);  
 }  
  
 @Test  
 void downloadPhoto() {  
 Assertions.*assertNotNull*(bookService.downloadPhoto(2L));  
 }  
}

