

Spring Testing. Junit и Mockito для написания тестов







Евгений Манько

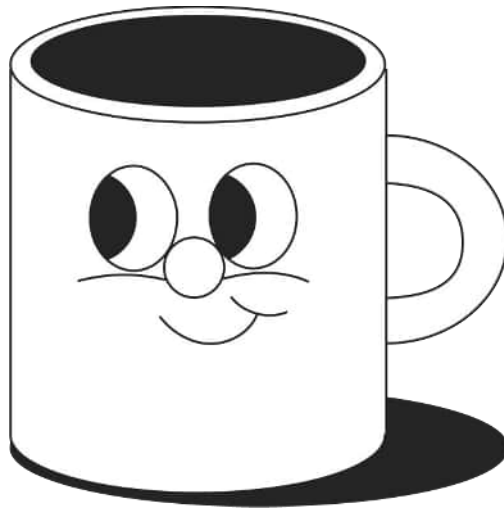
Java-разработчик, создатель данного курса

- ☀ Разрабатывал бэкенд для Яндекс, Тинькофф, МТС;
- ☀ Победитель грантового конкурса от «Росмолодежь»;
- ☀ Руководил IT-Департаментом «Студенты Москвы».






Что будет на уроке сегодня

-  Unit - тесты
-  Интеграционные тесты
-  Нагрузочное тестирование
-  Проект на Spring



Известные случаи багов

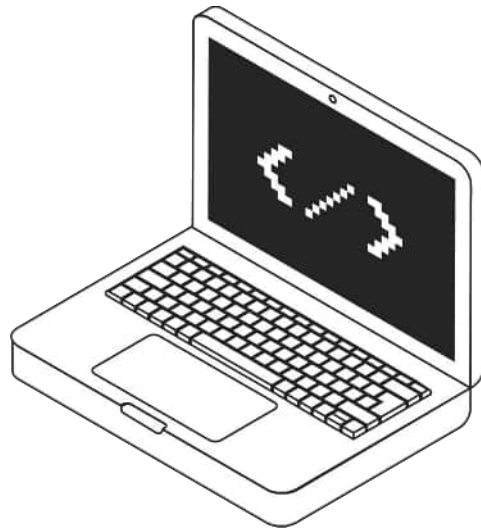
- 
Шаттл «Маринер-1»: В 1962 году НАСА потеряло космический корабль стоимостью 80 миллионов долларов из-за одной неверной точки в коде.
- 
Amazon Web Services, 2017: Один из инженеров AWS ввел неверную команду, пытаясь отладить систему.
- 
Knight Capital Group: Эта торговая компания потеряла 440 миллионов долларов за 45 минут из-за ошибки в своей торговой системе.





Виды тестов

1. Модульные тесты (Unit Tests)
2. Интеграционные тесты
3. Системные тесты
4. Тесты приемки
5. Нагрузочные тесты





Unit-тесты



Почему важны?

1. Быстрая обратная связь
2. Повышение уверенности
3. Упрощение изменений



Как писать?

1. Изолированность
2. 1 проверка на тест
3. Читаемость
4. ARR (Arrange, Act, Assert)

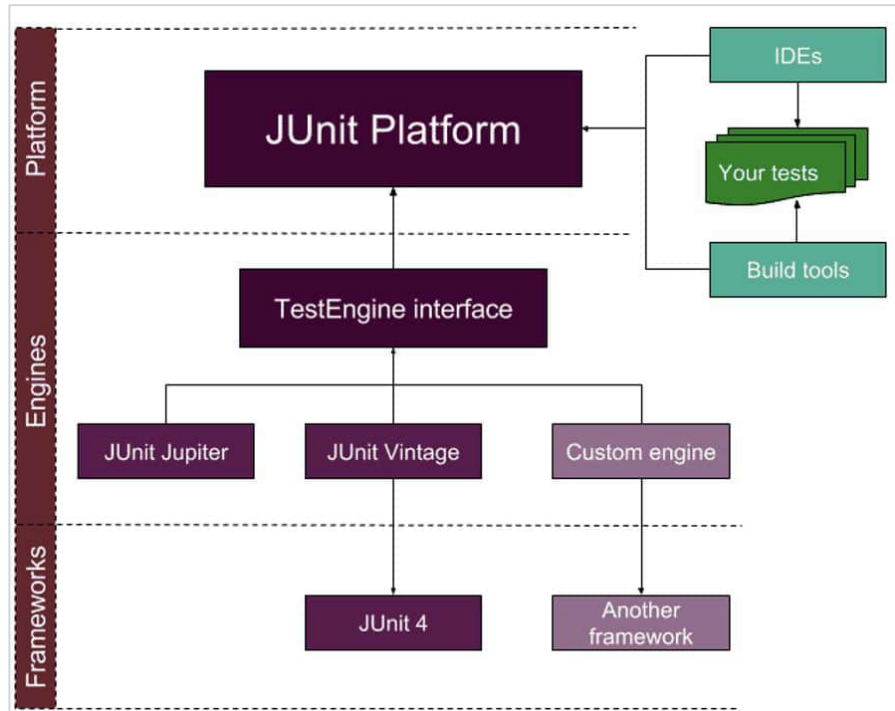


Unit-тесты

```
1 test function test_addition():
2     // Arrange
3     input1 = 2
4     input2 = 3
5     expected_result = 5
6
7     // Act
8     result = add(input1, input2)
9
10    // Assert
11    assert result == expected_result|
```



Unit-тесты в Java





Spring и Unit-тесты

1. Spring TestContext Framework
2. @SpringBootTest
3. Mocking с Mockito и @MockBean
4. @WebMvcTest и @DataJpaTest
5. @TestConfiguration
6. @BeforeAll и @AfterAll





Spring и Unit-тесты

```
1 @SpringBootTest
2 public class MyServiceTest {
3
4     @Autowired
5     private MyService myService;
6
7     @MockBean
8     private MyRepository myRepository;
9
10    @Test
11    public void testMyService() {
12        when(myRepository.findSomething()).thenReturn(someData);
13
14        // Некоторые проверки для myService
15    }
16 }
```



Spring и Unit-тесты

```
1 @TestConfiguration
2 public class MyTestConfiguration {
3     @Bean
4     public MyService myService() {
5         return new MyMockedService();
6     }
7 }
```

```
1 @SpringBootTest
2 public class MyTests {
3     @BeforeAll
4     public static void init() {
5         // Некоторая инициализация
6     }
7
8     @AfterAll
9     public static void cleanup() {
10        // Очистка после тестов
11    }
12 }
```



Интеграционные тесты



Какие аспекты они проверяют?

- Взаимодействие с базами данных
- Общение с внешними сервисами
- Взаимодействие между модулями



Сложности с:

- Настройка и поддержка
- Скорость выполнения
- Неустойчивость



Интеграционные тесты на Spring

1. Spring TestContext Framework
2. @SpringBootTest
3. Встроенная поддержка баз данных
4. MockMvc для веб-тестирования
5. Spring Boot Test Slices

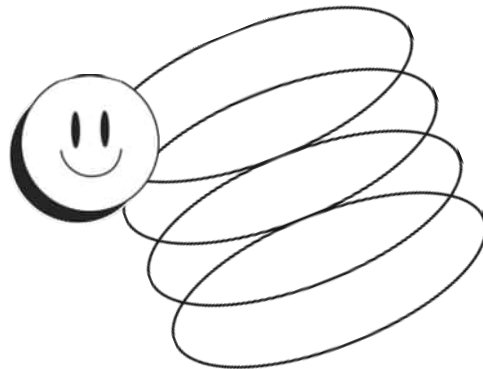




Нагрузочное тестирование

Зачем?

1. Производительность
2. Стабильность
3. Масштабируемость

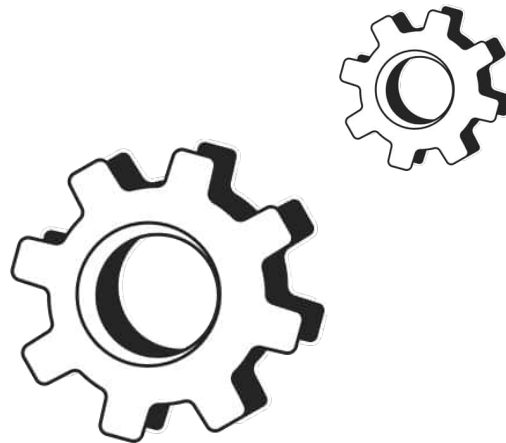




Нагрузочное тестирование

Как провести?

1. Определите ожидаемую нагрузку
2. Создайте тестовые сценарии
3. Подготовьте тестовое окружение
4. Выполните тесты
5. Анализируйте результаты





Нагрузочное тестирование в контексте Java





Нагрузочное тестирование с помощью JMeter

1. Установка JMeter
2. Создание тестового плана
3. Настройка запросов
4. Добавление слушателей
5. Запуск теста



Проект на Spring

1. Создание проекта

Перейдите на [Spring Initializr](#) и выберите следующие настройки:

- Project: Maven
- Language: Java
- Packaging: Jar
- Java: 17

Dependencies: — Spring Web — Spring Data JPA — H2 Database



Проект на Spring

2. Сущности

```
1 @Entity
2 public class Note {
3
4     @Id
5     @GeneratedValue(strategy = GenerationType.IDENTITY)
6     private Long id;
7
8     private String title;
9
10    @Lob
11    private String content;
12
13    // Геттеры, сеттеры и конструкторы
14 }
```



Проект на Spring

3. Репозиторий

```
1 @Repository
2 public interface NoteRepository extends JpaRepository<Note, Long> {}
```

4. Сервис

```
1 @Service
2 public class NoteService {
3
4     @Autowired
5     private NoteRepository noteRepository;
6
7     public List<Note> getAllNotes() {
8         return noteRepository.findAll();
9     }
10
11     public Note getNoteById(Long id) {
12         return noteRepository.findById(id).orElse(null);
13     }
14
15     public Note saveOrUpdate(Note note) {
16         return noteRepository.save(note);
17     }
18
19     public void deleteNote(Long id) {
20         noteRepository.deleteById(id);
21     }
22 }
```



Проект на Spring

5. Контроллер

```
1 @RestController
2 @RequestMapping("/api/notes")
3 public class NoteController {
4
5     @Autowired
6     private NoteService noteService;
7
8     @GetMapping
9     public List<Note> getAllNotes() {
10         return noteService.getAllNotes();
11     }
12
13     @GetMapping("/{id}")
14     public Note getNote(@PathVariable Long id) {
15         return noteService.getNoteById(id);
16     }
17
18     @PostMapping
19     public Note createNote(@RequestBody Note note) {
20         return noteService.saveOrUpdate(note);
21     }
22
23     @PutMapping("/{id}")
24     public Note updateNote(@PathVariable Long id, @RequestBody Note updatedNote) {
25         Note note = noteService.getNoteById(id);
26         note.setTitle(updatedNote.getTitle());
27         note.setContent(updatedNote.getContent());
28         return noteService.saveOrUpdate(note);
29     }
30
31     @DeleteMapping("/{id}")
32     public void deleteNote(@PathVariable Long id) {
33         noteService.deleteNote(id);
34     }
35 }
```



Проект на Spring

6. Настройка базы данных

Откройте application.properties и добавьте следующее:

```
1 spring.datasource.url=jdbc:h2:mem:notesdb
2 spring.datasource.driver-class-name=org.h2.Driver
3 spring.datasource.username=sa
4 spring.datasource.password=
5 spring.jpa.database-platform=org.hibernate.dialect.H2Dialect
6
7 spring.h2.console.enabled=true
```

7. Запуск

Теперь, когда у нас есть базовое приложение, вы можете запустить его, используя `mvn spring-boot:run` или из вашей IDE.



Юнит-тестирование нашего проекта

1. Подготовка зависимостей

```
1 <!-- JUnit -->
2 <dependency>
3     <groupId>org.springframework.boot</groupId>
4     <artifactId>spring-boot-starter-test</artifactId>
5     <scope>test</scope>
6 </dependency>
7
8 <!-- Mockito -->
9 <dependency>
10    <groupId>org.mockito</groupId>
11    <artifactId>mockito-core</artifactId>
12    <scope>test</scope>
13 </dependency>
```



Юнит-тестирование нашего проекта

2. Написание теста для NoteService

```
1 @RunWith(SpringRunner.class)
2 public class NoteServiceTest {
3
4     @InjectMocks
5     private NoteService noteService;
6
7     @Mock
8     private NoteRepository noteRepository;
9
10    @Before
11    public void setUp() {
12        MockitoAnnotations.initMocks(this);
13    }
14
15    @Test
16    public void getAllNotesTest() {
17        Note note = new Note();
18        note.setTitle("Test Title");
19        note.setContent("Test Content");
20
21        List<Note> expectedNotes = Collections.singletonList(note);
22
23        when(noteRepository.findAll()).thenReturn(expectedNotes);
24
25        List<Note> actualNotes = noteService.getAllNotes();
26
27        assertEquals(expectedNotes, actualNotes);
28    }
29
30    // ... Другие тесты для методов сервиса
31 }
```




Интеграционное тестирование нашего проекта

1. Настройка тестового окружения

```
1 <dependency>
2     <groupId>com.h2database</groupId>
3     <artifactId>h2</artifactId>
4     <scope>test</scope>
5 </dependency>|
```



Интеграционное тестирование нашего проекта

2. Написание теста для NoteService

```
1 @RunWith(SpringRunner.class)
2 @SpringBootTest
3 public class NoteServiceIntegrationTest {
4
5     @Autowired
6     private NoteService noteService;
7
8     @Autowired
9     private NoteRepository noteRepository;
10
11     @Before
12     public void setUp() {
13         // Очищаем базу данных перед каждым тестом
14         noteRepository.deleteAll();
15     }
16
17     @Test
18     public void getAllNotesIntegrationTest() {
19         Note note = new Note();
20         note.setTitle("Integration Test Title");
21         note.setContent("Integration Test Content");
22
23         noteRepository.save(note);
24
25         List<Note> notes = noteService.getAllNotes();
26         assertTrue(notes.size() > 0);
27         assertEquals(note.getTitle(), notes.get(0).getTitle());
28     }
29
30     // ... Другие интеграционные тесты
31 }
```



Нагрузочное тестирование нашего Spring-проекта

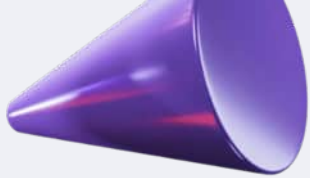
1. Установка JMeter.
2. Создание тестового плана.
3. Добавление и конфигурации HTTP Request.
4. Добавление Listener.
5. Запуск теста.
6. Анализ результатов.



Метрики нагрузочного тестирования

1. Throughput (Пропускная способность)
2. Response Time (Время ответа)
3. Error Rate (Процент ошибок)
4. Concurrent Users
(Конкурентные пользователи)
5. CPU/Memory Utilization
(Использование ЦПУ/Памяти)





Спасибо за внимание

