

CODELEAK.PL

by Rafał Borowiec

Fork me on GitHub

[BLOG](#)[SPRING](#)[AUTHOR](#)

SPRING BOOT TESTING WITH JUNIT 5

JUnit 5 (JUnit Jupiter) is around for quite some time already and it is equipped with tons of features. But surprisingly JUnit 5 it is not a default test library dependency when it comes to the Spring Boot Test Starter: it is *still* JUnit 4.12, released back in 2014. If you consider using JUnit 5 in you next Spring Boot based project then this blog post is for you. You will learn about the basic setup for Gradle and Maven based projects with examples of Spring Boot tests for different use cases.

Table of contents

- [Source code](#)
- [Setup the project from the ground up](#)
 - [Build with Gradle](#)
 - [Build with Maven](#)
 - [Use JUnit 5 in the test class](#)
 - [Run the test](#)
 - [Source code](#)
- [Sample application with a single REST controller](#)
 - [Source code](#)
- [Creating Spring Boot tests](#)
 - [Spring Boot test with web server running on random port](#)
 - [Spring Boot test with web server running on random port with mocked dependency](#)
 - [Spring Boot test with mocked MVC layer](#)
 - [Spring Boot test with mocked MVC layer and mocked dependency](#)
 - [Spring Boot test with mocked web layer](#)
 - [Spring Boot test with mocked web layer and mocked dependency](#)
 - [Run all tests](#)
- [References](#)
- [See also](#)

Source code

The source code for this article can be found on Github: <https://github.com/kolorobot/spring-boot-junit5>.

Setup the project from the ground up

For the project setup you will need JDK 11 or later and Gradle or Maven (depending on your preference). The easiest way to get started with Spring Boot is to use the Initializr at <https://start.spring.io>. The only dependencies to select is Spring Web. Testing dependencies (Spring Boot Starter Test) are always included, no matter what dependencies you use in the generated project.

Build with Gradle

The default project file for Gradle build (gradle.build) generated with Initializr:

```
plugins {  
    id 'org.springframework.boot' version '2.1.8.RELEASE'  
    id 'io.spring.dependency-management' version '1.0.8.RELEASE'  
    id 'java'  
}  
  
group = 'pl.codeleak.samples'  
version = '0.0.1-SNAPSHOT'  
sourceCompatibility = '11'  
  
repositories {  
    mavenCentral()  
}  
  
dependencies {  
    implementation 'org.springframework.boot:spring-boot-starter-  
testImplementation 'org.springframework.boot:spring-boot-starter-
```



To add JUnit 5 support we need to exclude the old JUnit 4 dependency and include JUnit 5 (JUnit Jupiter) dependency:

```
dependencies {  
    implementation 'org.springframework.boot:spring-boot-starter-
```

```

testImplementation('org.springframework.boot:spring-boot-starter-test')
    exclude group: 'junit', module: 'junit'
}
testCompile 'org.junit.jupiter:junit-jupiter:5.5.2'
}

test {
    useJUnitPlatform()
    testLogging {
        events "passed", "skipped", "failed"
    }
}

```

Build with Maven

The default project file for Maven build (pom.xml) generated with Initializr:

```

<?xml version="1.0" encoding="UTF-8"?>
<project>
    <modelVersion>4.0.0</modelVersion>
    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>2.1.8.RELEASE</version>
        <relativePath/> <!-- lookup parent from repository -->
    </parent>
    <groupId>pl.codeleak.samples</groupId>
    <artifactId>spring-boot-junit5</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <name>spring-boot-junit5</name>
    <description>Demo project for Spring Boot and JUnit 5</description>

    <properties>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
        <java.version>11</java.version>
    </properties>

    <dependencies>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>
    </dependencies>


```

```

    <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-test</artifactId>
      <scope>test</scope>
    </dependency>
  </dependencies>

  <build>
    <plugins>
      <plugin>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-maven-plugin</artifactId>
      </plugin>
    </plugins>
  </build>
</project>

```



To add JUnit 5 support we need to exclude the old JUnit 4 dependency and include JUnit 5 (JUnit Jupiter) dependency:

```

<properties>
  <junit.jupiter.version>5.5.2</junit.jupiter.version>
</properties>

<dependencies>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
  </dependency>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-test</artifactId>
    <scope>test</scope>
    <exclusions>
      <exclusion>
        <groupId>junit</groupId>
        <artifactId>junit</artifactId>
      </exclusion>
    </exclusions>
  </dependency>
  <dependency>
    <groupId>org.junit.jupiter</groupId>

```

```
<artifactId>junit-jupiter</artifactId>
<version>${junit.jupiter.version}</version>
<scope>test</scope>
</dependency>
</dependencies>
```

Use JUnit 5 in the test class

The test generated by the Initializr contains automatically generated JUnit 4 test. To apply JUnit 5 we need to change the imports and replace the JUnit 4 runner by the JUnit 5 extension. We can also make the class and the test method package protected:

```
import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.extension.ExtendWith;
import org.springframework.boot.test.context.SpringBootTest;
import org.springframework.test.context.junit.jupiter.SpringExtension;

@ExtendWith(SpringExtension.class)
@SpringBootTest
class SpringBootJUnit5ApplicationTests {

    @Test
    void contextLoads() {
    }

}
```

Tip: If you are new to JUnit 5 see my other posts about JUnit 5:

[https://blog.codeleak.pl/search/label/junit 5](https://blog.codeleak.pl/search/label/junit%205)

Run the test

We can run the test either with Maven Wrapper: `./mvnw clean test` or with Gradle Wrapper: `./gradlew clean test`.

Source code

Please consult [this commit](#) for the changes related to the project setup.

Sample application with a single REST controller

The sample application is containing a single REST controller with three endpoints:

- /tasks/{id}
- /tasks
- /tasks?title={title}

Each of the controller's method is calling internally JSONPlaceholder – fake online REST API for testing and prototyping.

The structure of the project files is as follows:

```
$ tree src/main/java
src/main/java
├── pl
│   ├── codeleak
│   │   └── samples
│   │       └── springbootjunit5
│   │           ├── SpringBootJunit5Application.java
│   │           ├── config
│   │           │   ├── JsonPlaceholderApiConfig.java
│   │           │   └── JsonPlaceholderApiConfigProperties.java
│   │           └── todo
│   │               ├── JsonPlaceholderTaskRepository.java
│   │               ├── Task.java
│   │               ├── TaskController.java
│   │               └── TaskRepository.java
```

It also have the following static resources:

```
$ tree src/main/resources/
src/main/resources/
├── application.properties
├── static
│   ├── error
│   │   └── 404.html
│   └── index.html
└── templates
```

The TaskController is delegating its work to the TaskRepository:

```
@RestController
class TaskController {

    private final TaskRepository taskRepository;

    TaskController(TaskRepository taskRepository) {
        this.taskRepository = taskRepository;
    }

    @GetMapping("/tasks/{id}")
    Task findOne(@PathVariable Integer id) {
        return taskRepository.findOne(id);
    }

    @GetMapping("/tasks")
    List<Task> findAll() {
        return taskRepository.findAll();
    }

    @GetMapping(value = "/tasks", params = "title")
    List<Task> findByTitle(String title) {
        return taskRepository.findByTitle(title);
    }
}
```

The TaskRepository is implemented by JsonPlaceholderTaskRepository that is using internally RestTemplate for calling JSONPlaceholder (<https://jsonplaceholder.typicode.com>) endpoint:

```
public class JsonPlaceholderTaskRepository implements TaskRepository {

    private final RestTemplate restTemplate;
    private final JsonPlaceholderApiConfigProperties properties;

    public JsonPlaceholderTaskRepository(RestTemplate restTemplate,
        JsonPlaceholderApiConfigProperties properties) {
        this.restTemplate = restTemplate;
        this.properties = properties;
    }

    @Override
    public Task findOne(Integer id) {
        return restTemplate
            .getForObject("/todos/{id}", Task.class, id);
    }
}
```

```
}

// other methods skipped for readability

}
```

The application is configured via `JsonPlaceholderApiConfig` that is using `JsonPlaceholderApiConfigProperties` to bind some sensible properties from `application.properties`:

```
@Configuration
@EnableConfigurationProperties(JsonPlaceholderApiConfigProperties)
public class JsonPlaceholderApiConfig {

    private final JsonPlaceholderApiConfigProperties properties;

    public JsonPlaceholderApiConfig(JsonPlaceholderApiConfigProperties properties) {
        this.properties = properties;
    }

    @Bean
    RestTemplate restTemplate() {
        return new RestTemplateBuilder()
            .rootUri(properties.getRootUri())
            .build();
    }

    @Bean
    TaskRepository taskRepository(RestTemplate restTemplate, JsonPlaceholderApiConfigProperties properties) {
        return new JsonPlaceholderTaskRepository(restTemplate, properties);
    }
}
```

The `application.properties` contain several properties related to the `JSONPlaceholder` endpoint configuration:


```
json-placeholder.root-uri=https://jsonplaceholder.typicode.com
json-placeholder.todo-find-all.sort=id
json-placeholder.todo-find-all.order=desc
json-placeholder.todo-find-all.limit=20
```

Read more about `@ConfigurationProperties` in this blog post:

<https://blog.codeleak.pl/2014/09/using-configurationproperties-in-spring.html>

Source code

Please consult [this commit](#) for the changes related to the source code of the application.

Creating Spring Boot tests

Spring Boot provides a number of utilities and annotations that support testing applications.

Different approaches can be used while creating the tests. Below you will find the most common cases for creating Spring Boot tests.

Spring Boot test with web server running on random port

```
@ExtendWith(SpringExtension.class)
@SpringBootTest(webEnvironment = SpringBootTest.WebEnvironment.RA
class TaskControllerIntegrationTest {

    @LocalServerPort
    private int port;

    @Autowired
    private TestRestTemplate restTemplate;

    @Test
    void findsTaskById() {
        // act
        var task = restTemplate.getForObject("http://localhost:"

        // assert
        assertThat(task)
            .extracting(Task::getId, Task::getTitle, Task::is
            .containsExactly(1, "delectus aut autem", false,
```

```
}  
}
```

Spring Boot test with web server running on random port with mocked dependency

```
@ExtendWith(SpringExtension.class)  
@SpringBootTest(webEnvironment = SpringBootTest.WebEnvironment.RA  
class TaskControllerIntegrationTestWithMockBeanTest {  
  
    @LocalServerPort  
    private int port;  
  
    @MockBean  
    private TaskRepository taskRepository;  
  
    @Autowired  
    private TestRestTemplate restTemplate;  
  
    @Test  
    void findsTaskById() {  
  
        // arrange  
        var taskToReturn = new Task();  
        taskToReturn.setId(1);  
        taskToReturn.setTitle("delectus aut autem");  
        taskToReturn.setCompleted(true);  
        taskToReturn.setUserId(1);  
  
        when(taskRepository.findOne(1)).thenReturn(taskToReturn);  
  
        // act  
        var task = restTemplate.getForObject("http://localhost:"  
  
        // assert  
        assertThat(task)  
            .extracting(Task::getId, Task::getTitle, Task::is  
            .containsExactly(1, "delectus aut autem", true, 1  
    }  
}
```

Spring Boot test with mocked MVC layer

```
@ExtendWith(SpringExtension.class)
@SpringBootTest
@AutoConfigureMockMvc
class TaskControllerMockMvcTest {

    @Autowired
    private MockMvc mockMvc;

    @Test
    void findsTaskById() throws Exception {
        mockMvc.perform(get("/tasks/1"))
            .andExpect(status().isOk())
            .andExpect(content().json("{\"id\":1,\"title\":\""))
    }
}
```



Spring Boot test with mocked MVC layer and mocked dependency

```
@ExtendWith(SpringExtension.class)
@SpringBootTest
@AutoConfigureMockMvc
class TaskControllerMockMvcWithMockBeanTest {

    @Autowired
    private MockMvc mockMvc;

    @MockBean
    private TaskRepository taskRepository;

    @Test
    void findsTaskById() throws Exception {
        // arrange
        var taskToReturn = new Task();
        taskToReturn.setId(1);
        taskToReturn.setTitle("delectus aut autem");
    }
}
```

```
taskToReturn.setCompleted(true);
taskToReturn.setUserId(1);

when(taskRepository.findOne(1)).thenReturn(taskToReturn);

// act and assert
mockMvc.perform(get("/tasks/1"))
    .andExpect(status().isOk())
    .andExpect(content().json("{\"id\":1,\"title\":\"\"}"))
}
}
```

Spring Boot test with mocked web layer

```
@ExtendWith(SpringExtension.class)
@WebMvcTest
@Import(JsonPlaceholderApiConfig.class)
class TaskControllerWebMvcTest {

    @Autowired
    private MockMvc mockMvc;

    @Test
    void findsTaskById() throws Exception {
        mockMvc.perform(get("/tasks/1"))
            .andExpect(status().isOk())
            .andExpect(content().json("{\"id\":1,\"title\":\"\"}"))
    }
}
```

Spring Boot test with mocked web layer and mocked dependency

```
@ExtendWith(SpringExtension.class)
@WebMvcTest
class TaskControllerWebMvcWithMockBeanTest {
```

```
@Autowired
private MockMvc mockMvc;

@MockBean
private TaskRepository taskRepository;

@Test
void findsTaskById() throws Exception {
    // arrange
    var taskToReturn = new Task();
    taskToReturn.setId(1);
    taskToReturn.setTitle("delectus aut autem");
    taskToReturn.setCompleted(true);
    taskToReturn.setUserId(1);

    when(taskRepository.findOne(1)).thenReturn(taskToReturn);

    // act and assert
    mockMvc.perform(get("/tasks/1"))
        .andExpect(status().isOk())
        .andExpect(content().json("{\"id\":1,\"title\":\""))
}
}
```

Run all tests

We can run all tests either with Maven Wrapper: `./mvnw clean test` or with Gradle Wrapper: `./gradlew clean test`.

The results of running the tests with Gradle:

```
$ ./gradlew clean test
```

```
> Task :test
```

```
pl.codeleak.samples.springbootjunit5.SpringBootJUnit5ApplicationT
```

```
pl.codeleak.samples.springbootjunit5.todo.TaskControllerWebMvcTes
```

```
pl.codeleak.samples.springbootjunit5.todo.TaskControllerIntegrati
```

```
pl.codeleak.samples.springbootjunit5.todo.TaskControllerWebMvcWit
```

```
pl.codeleak.samples.springbootjunit5.todo.TaskControllerIntegrati
```

```
pl.codeleak.samples.springbootjunit5.todo.TaskControllerMockMvcTe
```

```
pl.codeleak.samples.springbootjunit5.todo.TaskControllerMockMvcWi
```

BUILD SUCCESSFUL in 7s

5 actionable tasks: 5 executed



References

- <https://docs.spring.io/spring-boot/docs/2.1.8.RELEASE/reference/html/boot-features-testing.html>
- <https://spring.io/guides/gs/testing-web/>
- <https://github.com/spring-projects/spring-boot/issues/14736>

See also

- [https://blog.codeleak.pl/search/label/junit 5](https://blog.codeleak.pl/search/label/junit%205)
- <https://blog.codeleak.pl/2014/09/using-configurationproperties-in-spring.html>
- <https://blog.codeleak.pl/2015/03/spring-boot-integration-testing-with.html>
- <https://blog.codeleak.pl/2014/09/testing-mail-code-in-spring-boot.html>

POSTED IN [JUNIT 5](#), [SPRING BOOT](#), [TESTING](#) ON 11:55 PM BY [RAFAŁ BOROWIEC](#) | [LEAVE A COMMENT](#)


[Home](#)

[Older Post](#)

0 komentarze:

Post a Comment

Enter your comment...

 Comment as: javimartinalon: ▼

Sign out

Publish

Preview

☐ Notify me

Linki do tego posta

Create a Link

Subscribe to: Post Comments (Atom)



CLEANER PARAMETERIZED TESTS WITH JUNIT 5

The general idea of parameterized unit tests is to run the same test method for different data. Creating parameterized tests in JUnit 4 is ...



FOLLOW ME

[Subscribe to Codeleak.pl with Feedburner](#)

[Subscribe to Codeleak.pl by Email](#)

POPULAR POSTS

[Git bash in IntelliJ IDEA on Windows](#)

[Spring Boot – Configure Log Level in runtime using actuator endpoint](#)

[Different ways of validating @RequestBody in Spring MVC with @Valid annotation](#)

[8 ways of handling exceptions in JUnit. Which one to choose?](#)

[Spring Boot and Spring Data REST –
exposing repositories over REST](#)

CATEGORIES

[spring boot](#) (24) [unit testing](#) (21) [java](#) (20) [spring mvc](#) (16) [spring](#) (15) [spring 4](#) (13) [thymeleaf](#) (12) [tools](#) (12) [java 8](#) (8) [junit 5](#) (7) [links](#) (7) [intellij](#) (5) [practices](#) (5) [spring security](#) (5) [angular2](#) (4) [assertj](#) (4) [openshift](#) (4) [productivity](#) (4) [testing](#) (4) [jersey](#) (3) [selenium](#) (3) [jee](#) (2) [maven](#) (2) [eclipse](#) (1) [gradle](#) (1) [kanban](#) (1) [lombok](#) (1) [quartz](#) (1) [spring 5](#) (1) [spring data](#) (1)

BLOG ARCHIVE

▼ [2019](#) (5)

▼ [September](#) (2)

[Spring Boot testing with JUnit 5](#)

[JUnit 5 and Selenium – Setup the
project with Grad...](#)

► [April](#) (1)

► [March](#) (2)

► [2017](#) (13)

► [2016](#) (12)

► [2015](#) (26)

► [2014](#) (42)

► [2013](#) (16)

► [2012](#) (3)

► [2011](#) (9)

Follow @kolorobot

COPYRIGHT © 2019 CODELEAK.PL | POWERED BY BLOGGER
DESIGN BY THEME WEAVER | DISTRIBUTED BY BLOGGER TEMPLATES | BLOGGER THEME BY
PREMIUMBLOGGERTEMPLATES.COM