

Testing Spring web applications



What you'll learn

- Define tests using Spring Boot
- Run the test and push the test report to Xray
- Validate in Jira that the test results are available

Overview

[Spring Framework](#) is a well-known Java framework to build Java-based applications, supporting [IoC \(Inversion of Control\) principle](#).

[Spring Boot](#) provides an opinionated extension on top of Spring that aims to minimize configuration burden and ease the implementation of applications.

With Spring it's possible to create web applications, REST services, and more.

Prerequisites

For this example we will use the built-in testing facilities provided by Spring to test the application developed in Spring Boot

We will need:

- Java and Maven installed in your machine
- [xray-junit-extension](#) maven plugin, to take advantage of some annotations that allow us to embed additional information on the generated JUnit XML report (optional)
 - create a file to enable the generation of an enhanced JUnit XML report that Xray can take advantage of

test/resources/META-INF/services/org.junit.platform.launcher.TestExecutionListener

```
app.getxray.xray.junit.customjunitxml.EnhancedLegacyXmlReportGeneratingListener
```

- configure the new reporter to generate the report in specific file (e.g., reports/TEST-junit-jupiter.xml)

test/resources/xray-junit-extensions.properties

```
report_directory=reports
```

To start using Spring Boot please follow the [Quick Start Guide](#) documentation; you can also use [Spring initializr](#) to make a working skeleton of a project using Spring and its dependencies.

Usually, Spring applications have these layers:

1. web/presentation layer
 - a. controllers, exception handlers, filters, ...
2. service layer
 - a. services with business logic
3. persistence/data layer
 - a. JPA Repository, Entity
 - b. database

The target SUT is a web application implemented using Spring Boot, having a REST API to manage users and some controllers that return text acting like typical servlets.

Our Spring application provides:

- 3 controllers:
 - IndexController: that is used to return the text "Welcome to this amazing website!" whenever accessing the root page /
 - GreetingController: that is used to return a greeting message (e.g., "Hello, xxx!") based on an HTML template
 - UserRestController: that provides a REST API to manage users using several endpoints under the `/users` base URL
- 1 service:
 - UserService/UserServiceImpl: to perform business logic on users; in this case just as a small layer on top of the repository (UserRepository)

- 1 entity and 1 associated repository:
 - User: a persistable entity
 - UserRepository: a JPA repository of User objects

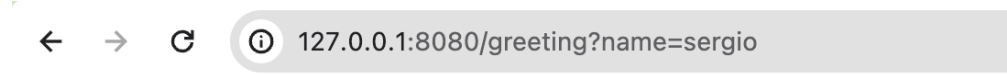
We can run our Spring application from the command line.

```
mvn spring-boot:run
```



← → ↻ ⓘ 127.0.0.1:8080

Welcome to this amazing website!



← → ↻ ⓘ 127.0.0.1:8080/greeting?name=sergio

Hello, sergio!

← → ↻ ⓘ 127.0.0.1:8080/api/users/1

```
{"id":1,"name":"Sergio Freire","username":"sergiofreire","password":"secret"}
```

← → ↻ ⓘ 127.0.0.1:8080/api/users

```
[{"id":1,"name":"Sergio Freire","username":"sergiofreire","password":"secret"}]
```

We'll implement JUnit 5 tests for all these layers:

- web:
 - we'll test the `IndexController` and `GreetingController` controllers
 - we'll test the REST API provided by the `UserRestController` controller
- service:
 - we will unit test the `UserServiceImpl`, avoiding usage of database, to test the logic of the service; we'll use `Mockito` to mock responses of the `UserRepository`
- data:
 - we'll test the `UserRepository` in isolation, without loading the web environment
 - an in-memory database (H2) will be used



Spring Boot supports [test slicing](#); the idea is to provide [slices](#) of the whole `ApplicationContext` by loading fewer components, thus providing efficiency. We'll see more about `@DataJpaTest` and `@WebMvcTest` ahead.

To test at the data layer, we can use `@DataJpaTest` as a test slice to test our `UserRepository` repository and the `User` entity.

- By default, tests annotated with `@DataJpaTest` are transactional and roll back at the end of each test. They also use an embedded in-memory database (H2).
- Can make use of `@TestEntityManager`: a test-friendly `EntityManager` that provides additional methods useful for testing

To test at the service layer, we won't need to boot the application; we can perform unit tests and mock dependency on the data layer.

To test at web layer we can follow different approaches by annotating the related test classes:

- `@SpringBootTest`
 - loads the full application; more resource intensive
 - web server port is injected using `@LocalServerPort`; a friendly REST client can be used by an injected `TestRestTemplate`
 - can use a specific database for testing purposes using `@AutoConfigureTestDatabase` and `@TestPropertySource`
- `@SpringBootTest(webEnvironment = WebEnvironment.MOCK, classes = ...)` + `@AutoConfigureMockMvc`
 - loads the full application except the web server itself
 - *Another useful approach is to not start the server at all but to test only the layer below that, where Spring handles the incoming HTTP request and hands it off to your controller. That way, almost all of the full stack is used, and your code will be called in exactly the same way as if it were processing a real HTTP request but without the cost of starting the server*
 - access to MVC framework is made using an injected reference to `MockMvc` using `@Autowired`
- `@WebMvcTest(xxx.class)`
 - loads a test slice focused just on the web layer, providing a simplified web environment
 - access to MVC framework is made using an injected reference to `MockMvc` using `@Autowired`
 - usually `@WebMvcTest` is used in combination with `@MockBean` or `@Import` to create any collaborators required by your `@Controller` beans.

Let's see some examples, precisely focused more on the web layer.

The following code snippet shows usage of `@WebMvcTest` to test a slice containing just the web layer. In this case we're testing the output of the root page.

Even though we don't have to use them, we'll also take advantage of 2 annotations provided by the `xray-junit-extensions` maven plugin to showcase additional features:

- **@XrayTest** to map the Junit test to an existing Test issue that already exists in Jira
- **@Requirement** to link the test to an existing requirement/story in Jira

IndexControllerMockedIT.java

```
package com.idera.xray.tutorials.springboot;

import static org.hamcrest.Matchers.equalTo;
import static org.springframework.test.web.servlet.result.MockMvcResultMatchers.content;
import static org.springframework.test.web.servlet.result.MockMvcResultMatchers.status;
import org.junit.jupiter.api.Test;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.autoconfigure.web.servlet.WebMvcTest;
import org.springframework.http.MediaType;
import org.springframework.test.web.servlet.MockMvc;
import org.springframework.test.web.servlet.request.MockMvcRequestBuilders;
import com.idera.xray.tutorials.springboot.boundary.IndexController;

// @SpringBootTest
// @AutoConfigureMockMvc; it is implied whenever @WebMvcTest is used

// @WebMvcTest annotation is used to test only the web layer of the application
// It disables full auto-configuration and instead apply only configuration relevant to MVC tests
@WebMvcTest(IndexController.class)
public class IndexControllerMockedIT {

    @Autowired
    private MockMvc mvc;

    @Test
    @XrayTest(key = "XT-676")
    @Requirement("XT-675")
    public void getWelcomeMessage() throws Exception {
        mvc.perform(MockMvcRequestBuilders.get("/").accept(MediaType.TEXT_PLAIN))
            .andExpect(status().isOk())
            .andExpect(content().string(equalTo("Welcome to this amazing website!")));
    }
}
```

The following code snippet loads the whole application using `@SpringBootTest` to test the REST API endpoints used to manage users.

UserRestControllerIT.java

```
package com.idera.xray.tutorials.springboot;

import org.json.JSONException;
import org.json.JSONObject;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.autoconfigure.jdbc.AutoConfigureTestDatabase;
import org.springframework.boot.test.context.SpringBootTest;
import org.springframework.boot.test.context.SpringBootTest.WebEnvironment;
import org.springframework.boot.test.web.client.TestRestTemplate;
import org.springframework.core.ParameterizedTypeReference;
import org.springframework.http.HttpMethod;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.util.UriComponentsBuilder;
import org.springframework.boot.test.web.server.LocalServerPort;
import com.idera.xray.tutorials.springboot.data.User;
import com.idera.xray.tutorials.springboot.data.UserRepository;
import app.getxray.xray.junit.customjunitxml.annotations.XrayTest;
import java.util.List;
import static org.assertj.core.api.Assertions.assertThat;

/* @SpringBootTest loads the full application, including the web server
 * @AutoConfigureTestDatabase is used to configure a test database instead of the application-defined database
 */
@SpringBootTest(webEnvironment = WebEnvironment.RANDOM_PORT)
@AutoConfigureTestDatabase
class UserRestControllerIT {

    @LocalServerPort
    int randomServerPort;

    @Autowired
    private TestRestTemplate restTemplate;

    @Autowired
    private UserRepository repository;

    User user1;

    @BeforeEach
    public void resetDb() {
        repository.deleteAll();
        user1 = repository.save(new User("Sergio Freire", "sergiofreire", "dummyspassword"));
    }

    @Test
    void createUserWithSuccess() {
        User john = new User("John Doe", "johndoe", "dummyspassword");
        ResponseEntity<User> entity = restTemplate.postForEntity("/api/users", john, User.class);

        List<User> foundUsers = repository.findAll();
        assertThat(foundUsers).extracting(User::getUsername).contains("johndoe");
    }

    @Test
    void dontCreateUserForInvalidData() {
        User john = new User("John Doe", "", "dummyspassword");
        ResponseEntity<User> response = restTemplate.postForEntity("/api/users", john, User.class);

        // ideally, the server shouldnt return 500, but 400 (bad request)
        assertThat(response.getStatusCode()).isEqualTo(HttpStatus.INTERNAL_SERVER_ERROR);

        List<User> found = repository.findAll();
    }
}
```

```

        assertThat(found).hasSize(1);
        assertThat(found).extracting(User::getName).doesNotContain("John Doe");
    }

    @Test
    void getUserWithSuccess() {
        String endpoint = UriComponentsBuilder.newInstance()
            .scheme("http")
            .host("127.0.0.1")
            .port(randomServerPort)
            .pathSegment("api", "users", user1.getId().toString() )
            .build()
            .toUriString();

        ResponseEntity<User> response = restTemplate.exchange(endpoint, HttpMethod.GET, null, new
ParameterizedTypeReference<User>() {
        });
        User user = response.getBody();

        assertThat(response.getStatusCode()).isEqualTo(HttpStatus.OK);
        assertThat(user1.equals(user)).isTrue();
    }

    @Test
    void getUserUnsuccess() throws JSONException {
        /*
        String endpoint = UriComponentsBuilder.newInstance()
            .scheme("http")
            .host("127.0.0.1")
            .port(randomServerPort)
            .pathSegment("api", "users", "-1" )
            .build()
            .toUriString();

        */

        ResponseEntity<JSONObject> response = restTemplate.exchange("/api/user/-1", HttpMethod.GET, null, new
ParameterizedTypeReference<JSONObject>() {
        });

        assertThat(response.getStatusCode()).isEqualTo(HttpStatus.NOT_FOUND);
    }

    @Test
    void listAllUsersWithSuccess() {
        createTempUser("Amanda James", "amanda", "dummyspassword");
        createTempUser("Robert Junior", "robert", "dummyspassword");

        ResponseEntity<List<User>> response = restTemplate
            .exchange("/api/users", HttpMethod.GET, null, new ParameterizedTypeReference<List<User>>() {
            });

        assertThat(response.getStatusCode()).isEqualTo(HttpStatus.OK);
        assertThat(response.getBody().extracting(User::getName).containsExactly("Sergio Freire", "Amanda
James", "Robert Junior"));
    }

    @Test
    void deleteUserWithSuccess() {
        ResponseEntity<User> response = restTemplate.exchange("/api/users/" + user1.getId(), HttpMethod.DELETE,
null, User.class);
        assertThat(response.getStatusCode()).isEqualTo(HttpStatus.OK);
        assertThat(response.getBody().getName()).isEqualTo("Sergio Freire");

        List<User> found = repository.findAll();
        assertThat(found).isEmpty();
    }

    @Test
    void deleteUserUnsuccess() {

```

```

        ResponseEntity<User> response = restTemplate.exchange("/api/users/" + (user1.getId()+2), HttpMethod.
DELETE, null, User.class);
        assertThat(response.getStatusCode()).isEqualTo(HttpStatus.NOT_FOUND);

        List<User> found = repository.findAll();
        assertThat(found).hasSize(1);
    }

    private void createTempUser(String name, String username, String password) {
        User user = new User(name, username, password);
        repository.saveAndFlush(user);
    }
}

```

In our case, we have tests that will be picked by *surefire* plugin and other ones that will be picked by *failsafe* plugin.

Once the code is implemented it can be executed with the following command:

```
mvn test failsafe:integration-test
```

The results are immediately available in the terminal.

```
[INFO] Tests run: 7, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.859 s -- in com.idera.xray.tutorials.springboot.UserRestControllerIT
[INFO] Running com.idera.xray.tutorials.springboot.IndexControllerMockedIT
2024-02-12T16:54:43.272Z INFO 42196 --- [main] t.c.s.AnnotationConfigContextLoaderUtils : Could not detect default configuration classes for test class [com.idera.xray.tutorials.springboot.IndexControllerMockedIT]: IndexControllerMockedIT does not declare any static, non-private, non-final, nested classes annotated with @Configuration.
2024-02-12T16:54:43.279Z INFO 42196 --- [main] .b.t.c.SpringBootTestContextBootstrapper : Found @SpringBootTestConfiguration com.idera.xray.tutorials.springboot.SpringBootApplication for test class com.idera.xray.tutorials.springboot.IndexControllerMockedIT
2024-02-12T16:54:43.282Z INFO 42196 --- [main] o.s.b.d.r.RestartApplicationListener : Restart disabled due to context in which it is running

  ____ _
 / ___ \ | |
| |___ \| |_| |
| _____|_| |
(W)      | |___|_| |
|   |   | |___|_| |
|   |   | |___|_| |
=====| |=====|_|_/
:: Spring Boot ::      (v3.2.2)

2024-02-12T16:54:43.299Z INFO 42196 --- [ring] main] c.i.x.t.s.IndexControllerMockedIT : Starting IndexControllerMockedIT using Java 21.0.1 with PID 42196 (started by sergio in /Users/sergio/experts/tutorial-springboot)
2024-02-12T16:54:43.299Z INFO 42196 --- [main] c.i.x.t.s.IndexControllerMockedIT : No active profile set, falling back to 1 default profile: "default"
2024-02-12T16:54:43.450Z INFO 42196 --- [main] o.s.b.t.m.w.SpringBootMockServletContext : Initializing Spring TestDispatcherServlet ''
2024-02-12T16:54:43.450Z INFO 42196 --- [main] o.s.t.web.servlet.TestDispatcherServlet : Initializing Servlet ''
2024-02-12T16:54:43.451Z INFO 42196 --- [main] o.s.t.web.servlet.TestDispatcherServlet : Completed initialization in 1 ms
2024-02-12T16:54:43.455Z INFO 42196 --- [main] c.i.x.t.s.IndexControllerMockedIT : Started IndexControllerMockedIT in 0.173 seconds (process running for 7.163)
[INFO] Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.192 s -- in com.idera.xray.tutorials.springboot.IndexControllerMockedIT
[INFO]
[INFO] Results:
[INFO]
[INFO] Tests run: 11, Failures: 0, Errors: 0, Skipped: 0
[INFO]
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 15.493 s
[INFO] Finished at: 2024-02-12T16:54:43Z
[INFO]
```

Ultimately this will lead to multiple JUnit XML reports (in `target/surefire-reports/` and `target/failsafe-reports/`, respectively).

If we use the `xray-junit-extensions` maven plugin, it will generate 1 JUnit XML report (i.e., in `reports/TEST-junit-jupiter.xml`) with all results of the last task executed (i.e., the integration tests ran by `failsafe` on the previous `mvn` command).

In this example, all tests have succeeded, as seen in the previous terminal screenshot. It generates the following JUnit XML report.

JUnit XML Report

```
<?xml version="1.0" encoding="UTF-8"?>
<testsuite name="JUnit Jupiter" tests="11" skipped="0" failures="0" errors="0" time="6"
  hostname="Sergios-MBP.lan" timestamp="2024-02-20T18:15:39">
  <properties>
    <property name="CONSOLE_LOG_CHARSET" value="UTF-8" />
    <property name="FILE_LOG_CHARSET" value="UTF-8" />
    <property name="user.country" value="PT" />
    <property name="user.timezone" value="Europe/Lisbon" />
  </properties>
```

```
<testcase name="getPersonalizedGreeting"
  classname="com.idera.xray.tutorials.springboot.GreetingControllerMockedIT" time="0"
  started-at="2024-02-20T18:15:37.51681" finished-at="2024-02-20T18:15:37.814122">
  <system-out><![CDATA[
unique-id: [engine:junit-jupiter]/[class:com.idera.xray.tutorials.springboot.GreetingControllerMockedIT]/
[method:getPersonalizedGreeting()]
display-name: getPersonalizedGreeting()
]]></system-out>
  <properties>
    <property name="_dummy_" value="" />
  </properties>
</testcase>
<testcase name="dontCreateUserForInvalidData"
  classname="com.idera.xray.tutorials.springboot.UserRestControllerIT" time="0"
  started-at="2024-02-20T18:15:38.375802" finished-at="2024-02-20T18:15:38.766131">
  <system-out><![CDATA[
unique-id: [engine:junit-jupiter]/[class:com.idera.xray.tutorials.springboot.UserRestControllerIT]/[method:
dontCreateUserForInvalidData()]
display-name: dontCreateUserForInvalidData()
]]></system-out>
  <properties>
    <property name="_dummy_" value="" />
  </properties>
</testcase>
<testcase name="getUserUnsuccess"
  classname="com.idera.xray.tutorials.springboot.UserRestControllerIT" time="0"
  started-at="2024-02-20T18:15:38.845142" finished-at="2024-02-20T18:15:38.868507">
  <system-out><![CDATA[
unique-id: [engine:junit-jupiter]/[class:com.idera.xray.tutorials.springboot.UserRestControllerIT]/[method:
getUserUnsuccess()]
display-name: getUserUnsuccess()
]]></system-out>
  <properties>
    <property name="_dummy_" value="" />
  </properties>
</testcase>
<testcase name="deleteUserWithSuccess"
  classname="com.idera.xray.tutorials.springboot.UserRestControllerIT" time="0"
  started-at="2024-02-20T18:15:38.804222" finished-at="2024-02-20T18:15:38.824641">
  <system-out><![CDATA[
unique-id: [engine:junit-jupiter]/[class:com.idera.xray.tutorials.springboot.UserRestControllerIT]/[method:
deleteUserWithSuccess()]
display-name: deleteUserWithSuccess()
]]></system-out>
  <properties>
    <property name="_dummy_" value="" />
  </properties>
</testcase>
<testcase name="createUserWithSuccess"
  classname="com.idera.xray.tutorials.springboot.UserRestControllerIT" time="0"
  started-at="2024-02-20T18:15:38.825412" finished-at="2024-02-20T18:15:38.844362">
  <system-out><![CDATA[
unique-id: [engine:junit-jupiter]/[class:com.idera.xray.tutorials.springboot.UserRestControllerIT]/[method:
createUserWithSuccess()]
display-name: createUserWithSuccess()
]]></system-out>
  <properties>
    <property name="_dummy_" value="" />
  </properties>
</testcase>
<testcase name="getDefaultGreeting"
  classname="com.idera.xray.tutorials.springboot.GreetingControllerMockedIT" time="0"
  started-at="2024-02-20T18:15:37.814867" finished-at="2024-02-20T18:15:37.818444">
  <system-out><![CDATA[
unique-id: [engine:junit-jupiter]/[class:com.idera.xray.tutorials.springboot.GreetingControllerMockedIT]/
[method:getDefaultGreeting()]
display-name: getDefaultGreeting()
]]></system-out>
  <properties>
    <property name="_dummy_" value="" />
  </properties>
```



```

</testcase>
<testcase name="getWelcomeMessage"
  classname="com.idera.xray.tutorials.springboot.IndexControllerMockedIT" time="0"
  started-at="2024-02-20T18:15:39.091011" finished-at="2024-02-20T18:15:39.097803">
  <system-out><![CDATA[
unique-id: [engine:junit-jupiter]/[class:com.idera.xray.tutorials.springboot.IndexControllerMockedIT]/[method:
getWelcomeMessage()]
display-name: getWelcomeMessage()
]]></system-out>
  <properties>
    <property name="requirements" value="XT-675" />
    <property name="test_key" value="XT-676" />
    <property name="_dummy_" value="" />
  </properties>
</testcase>
<testcase name="getUserWithSuccess"
  classname="com.idera.xray.tutorials.springboot.UserRestControllerIT" time="0"
  started-at="2024-02-20T18:15:38.767197" finished-at="2024-02-20T18:15:38.803133">
  <system-out><![CDATA[
unique-id: [engine:junit-jupiter]/[class:com.idera.xray.tutorials.springboot.UserRestControllerIT]/[method:
getUserWithSuccess()]
display-name: getUserWithSuccess()
]]></system-out>
  <properties>
    <property name="_dummy_" value="" />
  </properties>
</testcase>
<testcase name="listAllUsersWithSuccess"
  classname="com.idera.xray.tutorials.springboot.UserRestControllerIT" time="0"
  started-at="2024-02-20T18:15:38.888027" finished-at="2024-02-20T18:15:38.909762">
  <system-out><![CDATA[
unique-id: [engine:junit-jupiter]/[class:com.idera.xray.tutorials.springboot.UserRestControllerIT]/[method:
listAllUsersWithSuccess()]
display-name: listAllUsersWithSuccess()
]]></system-out>
  <properties>
    <property name="_dummy_" value="" />
  </properties>
</testcase>
<testcase name="getWelcomeMessage"
  classname="com.idera.xray.tutorials.springboot.IndexControllerIT" time="0"
  started-at="2024-02-20T18:15:36.488501" finished-at="2024-02-20T18:15:37.25418">
  <system-out><![CDATA[
unique-id: [engine:junit-jupiter]/[class:com.idera.xray.tutorials.springboot.IndexControllerIT]/[method:
getWelcomeMessage()]
display-name: getWelcomeMessage()
]]></system-out>
  <properties>
    <property name="_dummy_" value="" />
  </properties>
</testcase>
<testcase name="deleteUserUnsuccess"
  classname="com.idera.xray.tutorials.springboot.UserRestControllerIT" time="0"
  started-at="2024-02-20T18:15:38.869301" finished-at="2024-02-20T18:15:38.887141">
  <system-out><![CDATA[
unique-id: [engine:junit-jupiter]/[class:com.idera.xray.tutorials.springboot.UserRestControllerIT]/[method:
deleteUserUnsuccess()]
display-name: deleteUserUnsuccess()
]]></system-out>
  <properties>
    <property name="_dummy_" value="" />
  </properties>
</testcase>
<system-out><![CDATA[
unique-id: [engine:junit-jupiter]
display-name: JUnit Jupiter
]]></system-out>
</testsuite>

```

Integrating with Xray

Once we produced JUnit reports with the test results, it is a matter of importing those results into your Jira instance. This can be done by simply submitting automation results to Xray through the REST API, by using one of the available CI/CD plugins (e.g. for Jenkins), or using the Jira interface.

API

API

Once you have the report file available you can upload it to Xray through a request to the [REST API endpoint for JUnit](#), and for that the first step is to follow the instructions in [v1](#) or [v2](#) (depending on your usage) to obtain the token we will be using in the subsequent requests.

Authentication

The request made will look like:

```
curl -H "Content-Type: application/json" -X POST --data '{ "client_id": "CLIENTID", "client_secret": "CLIENTSECRET" }' https://xray.cloud.getxray.app/api/v2/authenticate
```

The response of this request will return the token to be used in the subsequent requests for authentication purposes.

JUnit XML results

Once you have the token we will use it in the API request with the definition of some common fields on the Test Execution, such as the target project, project version, etc.

```
curl -H "Content-Type: text/xml" -X POST -H "Authorization: Bearer $token" --data @"reports/TEST-junit-jupiter.xml" https://xray.cloud.getxray.app/api/v2/import/execution/junit?projectKey=XT&testPlanKey=XT-674
```

With this command we are creating a new Test Execution in the referred Test Plan with a generic summary and tests with a summary based on the test name. One of the tests was not auto-provisioned as it already existed beforehand and was referred in the the test code using the @XrayTest annotation.

Execution results [1708453141445]

Attach Create subtask Link issue Tests Risk assessment

Description
Add a description...

Tests
Add Tests Trigger Build View on board

Overall Execution Status

11 PASSED TOTAL TESTS: 11

Rank	Key	Summary	Test Type	Dataset	#Defects	TestRun Assignee	Priority	Status	Actions
1	XT-678	getPersonalizedGreeting	Generic		0	Sérgio Freire	=	PASSED	
2	XT-679	dontCreateUserForInvalidData	Generic		0	Sérgio Freire	=	PASSED	
3	XT-680	getUserUnsuccess	Generic		0	Sérgio Freire	=	PASSED	
4	XT-681	deleteUserWithSuccess	Generic		0	Sérgio Freire	=	PASSED	
5	XT-682	createUserWithSuccess	Generic		0	Sérgio Freire	=	PASSED	
6	XT-683	getDefaultGreeting	Generic		0	Sérgio Freire	=	PASSED	
7	XT-676	Test as a user, I can see a welcome message on the ...	Generic		0	Sérgio Freire	=	PASSED	
8	XT-684	getUserWithSuccess	Generic		0	Sérgio Freire	=	PASSED	
9	XT-685	listAllUsersWithSuccess	Generic		0	Sérgio Freire	=	PASSED	
10	XT-686	getWelcomeMessage	Generic		0	Sérgio Freire	=	PASSED	
11	XT-687	deleteUserUnsuccess	Generic		0	Sérgio Freire	=	PASSED	

Prev 1 Next Total 11 issues

Jira UI

Jira UI

1

Create a Test Execution linked to the Test Plan that you have.

Projects / Xray Tutorials / Add epic / XT-674

tutorial-spring

Attach Create subtask Link issue Tests Risk assessment

Description
Add a description...

Tests
Tests Test Executions

Add Tests Create Test Execution Trigger Build

Overall Execution
All tests...
With status...

11 PASSED

2

Fill in the necessary fields and press "Create".

Create planned Test Execution

Project

Xray Tutorials

Summary

Test Execution for Test Plan XT-674

Assignee

Sérgio Freire

Choose a user to assign the Test Execution

Select...

File Version/s

Select...

Test Environment

Select...

Go to Test Execution

Create

Cancel

3

Open the Test Execution and import the JUnit report

Projects / Xray Tutorials / Add epic / XT-688

Test Execution for Test Plan XT-674

Attach Create subtask Link issue Tests Risk assessment

Description

Add a description...

Tests

Add Tests Trigger Build View on board

Overall Execution Status

11 TO DO TOTAL TESTS: 11

Only My Test Runs Filters

Rank	Key	Summary	Test Type	Dataset	#Defects	TestRun Assignee	Priority	Status	Actions
1	XT-678	getPersonalizedGreeting	Generic		0	Sérgio Freire		TO DO	
2	XT-679	dontCreateUserForInvalidData	Generic		0	Sérgio Freire		TO DO	
3	XT-680	getUserUnsuccess	Generic		0	Sérgio Freire		TO DO	
4	XT-681	deleteUserWithSuccess	Generic		0	Sérgio Freire		TO DO	
5	XT-682	createUserWithSuccess	Generic		0	Sérgio Freire		TO DO	
6	XT-683	getDefaultGreeting	Generic		0	Sérgio Freire		TO DO	
7	XT-676	Test as a user, I can see a welcome message on the ...	Manual		0	Sérgio Freire		TO DO	
8	XT-684	getUserWithSuccess	Generic		0	Sérgio Freire		TO DO	
9	XT-685	testAllUsersWithSuccess	Generic		0	Sérgio Freire		TO DO	

Log work

Add flag

Configure jmx custom fields

Xporter

Xray - Generate Cucumber Feature files

Xray - Import Execution Results

Xray - Document Generator

Connect Slack channel

Convert to Subtask

Move

Clone

Delete

Find your field

Actions menu

Print

Export XML

Export Word

Take a tour

Find out more

4

Choose the results file and press "Submit"

Projects / Xray Tutorials / Add epic / XT-688

Test Execution

Import Execution Results

Attach

Description

Add a description

Tests

Add Tests

Overall Execution Status

11 TO DO TOTAL TESTS:

Choose file

TEST-junit-jupiter.xml

The file with the execution results for the Test Execution.

Submit

Cancel

5

The Test Execution is now updated with the test results imported.

Projects / [Xray Tutorials](#) / [Add epic](#) / [XT-688](#)

Test Execution for Test Plan XT-674

[Attach](#) [Create subtask](#) [Link issue](#) [Tests](#) [Risk assessment](#) [...](#)

Description

Add a description...

Tests

[Add Tests](#) [Trigger Build](#)

[View on board](#)

Overall Execution Status

11 PASSED

TOTAL TESTS: 11

Only My Test Runs										Filters	100	Columns
Rank	Key	Summary	Test Type	Dataset	#Defects	TestRun Assignee	Priority	Status	Actions			
<input type="checkbox"/>	1	XT-678	getPersonalizedGreeting	Generic	<div></div>	0	Sérgio Freire	<div></div>	<div></div>	PASSED	<div></div>	...
<input type="checkbox"/>	2	XT-679	dontCreateUserForInvalidData	Generic	<div></div>	0	Sérgio Freire	<div></div>	<div></div>	PASSED	<div></div>	...
<input type="checkbox"/>	3	XT-680	getUserUnsuccess	Generic	<div></div>	0	Sérgio Freire	<div></div>	<div></div>	PASSED	<div></div>	...
<input type="checkbox"/>	4	XT-681	deleteUserWithSuccess	Generic	<div></div>	0	Sérgio Freire	<div></div>	<div></div>	PASSED	<div></div>	...
<input type="checkbox"/>	5	XT-682	createUserWithSuccess	Generic	<div></div>	0	Sérgio Freire	<div></div>	<div></div>	PASSED	<div></div>	...
<input type="checkbox"/>	6	XT-683	getDefaultGreeting	Generic	<div></div>	0	Sérgio Freire	<div></div>	<div></div>	PASSED	<div></div>	...
<input type="checkbox"/>	7	XT-676	Test as a user, I can see a welcome message on the ...	Generic	<div></div>	0	Sérgio Freire	<div></div>	<div></div>	PASSED	<div></div>	...
<input type="checkbox"/>	8	XT-684	getUserWithSuccess	Generic	<div></div>	0	Sérgio Freire	<div></div>	<div></div>	PASSED	<div></div>	...
<input type="checkbox"/>	9	XT-685	listAllUsersWithSuccess	Generic	<div></div>	0	Sérgio Freire	<div></div>	<div></div>	PASSED	<div></div>	...
<input type="checkbox"/>	10	XT-686	getWelcomeMessage	Generic	<div></div>	0	Sérgio Freire	<div></div>	<div></div>	PASSED	<div></div>	...
<input type="checkbox"/>	11	XT-687	deleteUserUnsuccess	Generic	<div></div>	0	Sérgio Freire	<div></div>	<div></div>	PASSED	<div></div>	...
Prev 1 Next												
Total 11 issues												

Tests implemented using JUnit will have a corresponding Test entity in Xray. Once results are uploaded, Test issues corresponding to the tests are auto-provisioned (e.g., XT-678), unless they already exist; in our case, we have explicitly mentioned an existing Test issue (XT-676) from one of the tests.

Projects /  Xray Tutorials /  Add epic /  XT-678

getPersonalizedGreeting

 Attach  Create subtask  Link issue   Test details  Risk assessment 


Description

Add a description...

Test details

 Test details  Preconditions  Test Sets  Test Plans  Test Runs

Test Type

Generic 

Definition

com.idera.xray.tutorials.springboot.GreetingControllerMockedIT.getPersonalizedGreeting

Projects /  Xray Tutorials /  Add epic /  XT-676

Test as a user, I can see a welcome message on the root

 Attach  Create subtask  Link issue   Test details  Risk assessment 

Description

Add a description...

Linked issues

tests

 **XT-675** As a user, I can see a welcome message

As we have annotated one of the tests with @Requirement (provided by the [xray-junit-extensions](#) project), the linkage/coverage of the user story (XT-675) is made whenever importing the results.

Xray uses a concatenation of the suite name and the test name as the unique identifier for the test.

In Xray, results are stored in a Test Execution, usually a new one. The Test Execution contains a Test Run per each test that was executed.

Detailed results, including logs and exceptions reported during execution of the test, can be seen on the execution screen details of each Test Run, accessible through the execution details as we can see here:

Projects / Xray Tutorials / Add epic / XT-688

Test Execution for Test Plan XT-674

Attach Create subtask Link issue Tests Risk assessment

Description

Add a description...

Tests

Add Tests Trigger Build View on board

Overall Execution Status

11 PASSED TOTAL TESTS: 11

Only My Test Runs Filters 100 Columns

Rank	Key	Summary	Test Type	Dataset	#Defects	TestRun Assignee	Priority	Status	Actions
1	XT-678	getPersonalizedGreeting	Generic		0	Sérgio Freire		PASSED	
2	XT-679	dontCreateUserForInvalidData	Generic		0	Sérgio Freire		PASSED	
3	XT-680	getUserUnsuccess	Generic		0	Sérgio Freire		PASSED	
4	XT-681	deleteUserWithSuccess	Generic		0	Sérgio Freire		PASSED	
5	XT-682	createUserWithSuccess	Generic		0	Sérgio Freire		PASSED	

Projects / Xray Tutorials / Test Plans / XT-674 / Test Executions / XT-688 / Test: XT-678

Test 1 of 11

getPersonalizedGreeting

Execute with Exploratory App Dataset Import Execution Results

Execution Status PASSED Timer 00:00:00 Started On 20/Feb/2024 06:15 PM Assignee Sérgio Freire Test Version v1

Finished On 20/Feb/2024 06:15 PM Executed By - Revision - Test Environments -

Findings

Test details

Results

Context	Output	Duration	Status
TestSuite JUnit Jupiter	-	-	PASSED

Activity

As we linked one of the Tests to an existing Story (using the @Requirement annotation provided by the xray-junit-extensions project), on the Story issue screen we can track coverage having in mind the result of that test. We could link other Tests on the code or later on in Xray, by adding them using the "Add Tests" option.

Projects / Xray Tutorials / Add epic / XT-675

As a user, I can see a welcome message

Attach Create subtask Link issue Test Coverage Risk assessment

Description

Add a description...

Linked issues

is tested by

XT-676 Test as a user, I can see a welcome message on the root TO DO

Test Coverage

Add Tests Execute

Analysis & Scope

Scope: Latest Final Status OK

Status	Key	Summary	Test Status
TO DO	XT-676	Test as a user, I can see a welcome message on the ...	PASSED

Prev 1 Next

Tips

- after results are imported, in Jira Tests can be linked to existing requirements/user stories, so you can track the impact on their coverage; in this tutorial we've shown also how to link the test to a story right from the code.
- results from multiple builds can be linked to an existing Test Plan, to facilitate the analysis of test result trends across builds.
- results can be associated with a Test Environment, in case you want to analyze coverage and test results by that environment later on. A Test Environment can be a testing stage (e.g. dev, staging, preprod, prod) or a identifier of the device/application used to interact with the system (e.g. browser, mobile OS).

References

- [Spring project](#)
- [Spring Framework](#)
- [Spring Boot](#)
- [Spring guides](#)
- [Spring Testing](#)
- [article on Testing the web layer](#)
- [xray-junit-extensions](#)