

# Quarkus - Security Testing

This document describes how to test Quarkus Security.

## Configuring User Information

You can use [quarkus-elytron-security-properties-file](#) for testing security. This supports both embedding user info in `application.properties` and standalone properties files.

For example, the following configuration will allow for configuring the users in both the production where OAuth2 is required and development modes using [Configuration Profiles](#).

```
# Configure embedded authentication
%dev.quarkus.security.users.embedded.enabled=true
%dev.quarkus.security.users.embedded.plain-text=true
%dev.quarkus.security.users.embedded.users.scott=reader
%dev.quarkus.security.users.embedded.users.stuart=writer
%dev.quarkus.security.users.embedded.roles.scott=READER
%dev.quarkus.security.users.embedded.roles.stuart=READER,WRITER

# Configure OAuth2
quarkus.oauth2.enabled=true
%dev.quarkus.oauth2.enabled=false
quarkus.oauth2.client-id=client-id
quarkus.oauth2.client-secret=client-secret
quarkus.oauth2.introspection-url=http://host:port/introspect
```

## Test Security Extension

Quarkus provides explicit support for testing with different users, and with the security subsystem disabled. To use this you must include the `quarkus-test-security` artifact:

```
<dependency>
  <groupId>io.quarkus</groupId>
  <artifactId>quarkus-test-security</artifactId>
  <scope>test</scope>
</dependency>
```

This artifact provides the `io.quarkus.test.security.TestSecurity` annotation, that can be applied to test methods and test classes to control the security context that the test is run with. This allows you to do two things, you can disable authorization so tests can access secured endpoints without needing to be authenticated, and you can specify the identity that you want the tests to run under.

A test that runs with authorization disabled can just set the enabled property to false:

```
@Test
@TestSecurity(authorizationEnabled = false)
void someTestMethod() {
    ...
}
```

This will disable all access checks, which allows the test to access secured endpoints without needing to authenticate.

You can also use this to configure the current user that the test will run as:

```
@Test
@TestSecurity(user = "testUser", roles = {"admin", "user"})
void someTestMethod() {
    ...
}
```

This will run the test with an identity with the given username and roles. Note that these can be combined, so you can disable authorization while also providing an identity to run the test under, which can be useful if the endpoint expects an identity to be present.



The feature is only available for `@QuarkusTest` and will **not** work on a `@NativeImageTest`.

## Mixing security tests

If it becomes necessary to test security features using both `@TestSecurity` and Basic Auth (which is the fallback auth mechanism when none is defined), then Basic Auth needs to be enabled explicitly, for example by setting `quarkus.http.auth.basic=true` or `%test.quarkus.http.auth.basic=true`.

## Use Wiremock for Integration Testing

You can also use Wiremock to mock the authorization OAuth2 and OIDC services: See [OAuth2 Integration testing](#) for more details.

## References

- [Quarkus Security](#)