Quarkus Testing

Quarkus implements a set of functionalities to test Quarkus applications in an easy way as well as a tight integration with the REST Assured framework to write black-box tests.
This cheat sheet covers how to write component/integration tests in Quarkus.

CREATING THE PROJECT

```
mvn "io.quarkus:quarkus-maven-plugin:1.5.0.Final:create" \
  -DprojectGroupId="org.acme" \
  -DprojectArtifactId="greeting" \
  -DprojectVersion="1.0-SNAPSHOT" \
  -DclassName="org.acme.GreetingResource" \
  -Dpath="/hello"
```

Tip You can generate the project in https://code.quarkus.io/

TESTING

Quarkus archetype adds test dependencies with JUnit 5 and REST-Assured library to test REST endpoints.

```
@QuarkusTest
public class GreetingResourceTest {

    @Test
    public void testHelloEndpoint() {
        given()
          .when().get("/hello")
          .then()
             .statusCode(200)
             .body(is("hello"));
    }
}
```

By default test HTTP/S port is `8081` and `8444`. They can be overridden by setting the next properties:

```
quarkus.http.test-port=9090
quarkus.http.test-ssl-port=9091
```

If static resource is served, the URL can be injected in the test:

```
@TestHTTPResource("index.html")
URL url;
```

STUBBING

To provide an alternative implementation of an interface, you need to annotate the alternative service with `@io.quarkus.test.Mock` annotation.

```
@Mock
@ApplicationScoped
public class StubbedExternalService extends ExternalService {}

@Inject
ExternalService service; // (1)
```

1. Service is an instance of `StubbedExternalService`.

The alternative implementation overrides the real service for **all** test classes.

MOCK

You can also create mocks of your services with Mockito. Add the following dependency: `io.quarkus:quarkus-junit5-mockito`.

```
@InjectMock
GreetingService greetingService;

@BeforeEach
public void setup() {
    Mockito.when(greetingService.greet()).thenReturn("Hi");
}

@Path("/hello")
public class ExampleResource {

    @Inject
    GreetingService greetingService; // (1)
}
```

1. Mocked service.

Mocks are scoped to test class so they are only valid in the current class.
Spies are also supported by using `@InjectSpy`.

```
@InjectSpy
GreetingService greetingService;

Mockito.verify(greetingService, Mockito.times(1)).greet();
```

INTERCEPTORS

Since classes are in fact full CDI beans, you can apply CDI interceptors or create meta-annotations:

```
@QuarkusTest
@Stereotype
@Transactional
@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.TYPE)
public @interface TransactionalQuarkusTest {}

@TransactionalQuarkusTest
public class TestStereotypeTestCase {}
```

MicroProfile REST Client

To Mock REST Client, you need to define the REST Client interface with `@ApplicationScope` scope to be able to Mock it:

```
@ApplicationScoped
@RegisterRestClient
public interface GreetingService {
}

@InjectMock
@RestClient
GreetingService greetingService;

Mockito.when(greetingService.hello()).thenReturn("hello from mockito");
```

ACTIVE RECORD PATTERN WITH PANACHE

When implementing the active record pattern in Panache, there are some methods that are `static` and this makes the mocking process a bit complex. To avoid this complexity, Quarkus provides a special `PanacheMock` class that can be used to mock entities with static methods:

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

```
@Test
public void testPanacheMocking() {
    PanacheMock.mock(Person.class);

    Mockito.when(Person.count()).thenReturn(23l);
    Assertions.assertEquals(23, Person.count());
    PanacheMock.verify(Person.class, Mockito.times(1)).count();
}
```

QUARKUS TEST RESOURCE

You can execute some logic before the first test run and execute some logic at the end of the test suite.
You need to create a class implementing `QuarkusTestResourceLifecycleManager` interface and register it in the test via `@QuarkusTestResource` annotation.

```
public class MyCustomTestResource
    implements QuarkusTestResourceLifecycleManager {

    @Override
    public Map<String, String> start() {
        // return system properties that
        // are set before running tests
        return Collections.emptyMap();
    }

    @Override
    public void stop() {
    }

    // optional
    public void init(Map<String, String> initArgs) {} // (1)

    // optional
    @Override
    public void inject(Object testInstance) {}

    // optional
    @Override
    public int order() {
        return 0;
    }
}
```

1. Args are taken from ` QuarkusTestResource(initArgs)`.

Important Returning new system properties implies that if you run the tests in parallel, you need to run them in different JVMs.

And the registration of the test resource:

```
@QuarkusTestResource(MyCustomTestResource.class)
public class MyTest {}
```

Provided Test Resources

Quarkus provides some test resources implementations:

H2

Dependency: `io.quarkus:quarkus-test-h2` Registration: `@QuarkusTestResource(H2DatabaseTestResource.class)`.

Derby

Dependency: `io.quarkus:quarkus-test-derby` Registration: `@QuarkusTestResource(DerbyDatabaseTestResource.class)`

Artemis

Dependency: `io.quarkus:quarkus-test-artemis` Registration: `@QuarkusTestResource(ArtemisTestResource.class)`

LDAP

Dependency: `io.quarkus:quarkus-test-ldap` Registration: `@QuarkusTestResource(LdapServerTestResource.class)`
You can populate LDAP entries by creating a `quarkus-io.ldif` file at the root of the classpath.

Vault

Dependency: `io.quarkus:quarkus-test-vault` Registration: `@QuarkusTestResource(VaultTestLifecycleManager.class)`

Amazon Lambda

Dependency: `io.quarkus:quarkus-test-amazon-lambda` Registration: `@QuarkusTestResource(LambdaResourceManager.class)`

Kubernetes Mock Server

Dependency: `io.quarkus:quarkus-test-kubernetes-client` Registration:

```
@QuarkusTestResource(KubernetesMockServerTestResource.class)
    @MockServer
    private KubernetesMockServer mockServer;

    @Test
    public void test() {
        final Pod pod1 = ...
        mockServer
            .expect()
            .get()
            .withPath("/api/v1/namespaces/test/pods")
            .andReturn(200,
                new PodListBuilder()
                ...
    }
```

NATIVE TESTING

To test native executables annotate the test with `@NativeImageTest`.

ABOUT PROFILES

Quarkus allows you to have multiple configurations in the same file (`application.properties`).
The syntax for this is `%{profile}.config.key=value`.
`test` profile is used when tests are executed.

```
greeting.message=This is in Production
%test.greeting.message=This is a Test
```

AMAZON LAMBDAS

You can write tests for Amazon Lambdas:

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

```
@Test
public void testLambda() {
    MyInput in = new MyInput();
    in.setGreeting("Hello");
    in.setName("Stu");
```

```
    MyOutput out = io.quarkus.amazon.lambda.LambdaClient.invoke(MyOutput.class, in);
}
```

QUARKUS EMAIL

Quarkus offers an extension to send emails. A property is provided to send emails to a mock instance instead of using a real SMTP server.
If `quarkus.mailer.mock` is set to true, which is the default value in `dev` and `test` profiles, you can inject `MockMailbox` to get the sent messages.

```
@Inject
MockMailbox mailbox;

@BeforeEach
void init() {
    mailbox.clear();
}

List<Mail> sent = mailbox
            .getMessagesSentTo("to@acme.org");
```