Integration Testing

Effective Programming in Scala

Integration Testing

Testing a part of a system in isolation is good, but new problems can arise when we assemble the parts together.

Integration testing is the practice of testing a complete system, with no stubs or mocks.

Challenges of Integration Testing

What makes integration testing complex is that you need to be able to programmatically set up the entire stack (or significant parts of it) to a specific state.

This sometimes includes setting up external services your program depends on, such as a database.

Test-Local Resources With MUnit (1)

MUnit lets you set up and shut down a resource for the lifetime of a single test. For instance, to set up an HTTP server:

```
class HttpServerSuite extends munit.FunSuite:
  val withHttpServer = FunFixture[HttpServer](
    setup = test => {
      val httpServer = HttpServer()
      httpServer.start(8888)
      httpServer
    },
    teardown = httpServer => httpServer.stop()
  )
```

Test-Local Resources With MUnit (2)

```
class HttpServerSuite extends munit.FunSuite:

  val withHttpServer = FunFixture[HttpServer](
    setup = test => {
      val httpServer = HttpServer()
      httpServer.start(8888)
      httpServer
    },
    teardown = httpServer => httpServer.stop()
  )

  withHttpServer.test("server is running") { httpServer =>
    // Perform HTTP request here
  }
```

Suite-Local Resources With MUnit (1)

Sometimes, it's considerably more performant to set up the stack once at the beginning, then run all the tests, and eventually shut down the system.

We achieve that by overriding the methods `beforeAll` and `afterAll` of the `FunSuite` class:

```
class HttpServerSuite extends munit.FunSuite:

  val httpServer = HttpServer()
  override def beforeAll(): Unit = httpServer.start(8888)
  override def afterAll(): Unit  = httpServer.stop()
```

Suite-Local Resources With MUnit (2)

```
class HttpServerSuite extends munit.FunSuite:

  val httpServer = HttpServer()
  override def beforeAll(): Unit = httpServer.start(8888)
  override def afterAll(): Unit  = httpServer.stop()

  test("server is running") {
    // Perform HTTP request here
  }

  test("another test on the HTTP server") {
    // ...
  }
```

Summary

With integration tests, the system under test is closer to what is deployed to production.

On the other hand, such tests can be heavier to write, or can take a long time to complete.