In automated testing, we have 3 types of tests: unit tests, integration tests and end-to-end tests. A unit test is what I showed you

earlier in this section. You test a unit of the application without its external dependencies such as files, databases, message queues, web services and so on.

Now, note this keyword here: external dependencies.

This is really important, as you'll find out shortly. So, these unit tests exercise your code without any external dependencies, they are cheap to write and they execute fast. So, you can run hundreds of them in just a few seconds, and this way you can verify that each building block

in our application is working as expected. However, since you're not testing these classes or components with their external dependencies, you can't get a lot of confidence in the reliability of your application. So, that's when integration tests come to the rescue. An integration test, tests a class or a component with its external dependencies.

So, it tests the integration of your application code with these concrete dependencies like files, databases and so on. Again, note that I'm emphasizing the word "external" here. These tests, take longer to execute because they often involve reading or writing to a database, but they give us more confidence in the health of our application.

Now, traditionally, an integration test is defined as a test that takes a few units or classes and test their behavior as a whole.

So, based on this definition, if you test 2 classes together some people believe you're writing an integration test and not a unit test, even if none of these classes talk to an external resource like a database. Chances are you've heard this definition before.

Now I'll tell you what. This definition is a great recipe for writing fragile tests that are coupled to your implementation detail. So, as you change the implementation of your classes, these tests are gonna break and you'll end up wasting a lot of time fixing them. Not only won't they give you any values, but they actually slow you down! I'm gonna show you an example of this later in the course when I talk about fakes and mocks. If you've done unit testing before and failed, chances are you followed this definition! So, once again, a unit test tests a class or multiple classes with out their external dependencies. They test a unit of work.

An integration test tests a class or multiple classes with their external dependencies.

You also have another type of test called end-to-end test that drives an application through its user-interface. There are specific tools built for creating end-to-end tests. One popular tool that you might have heard of is Selenium, which allows us to record the interaction of a user with our application and then play it back and check if the application is returning the right result or not. These tests give you the greatest amount of confidence about the health of your application but they have 2 big problems.

The first problem is that they are very slow.

Because they require launching the application and testing it through the UI.

So, every test is going to launch the application, potentially

login, navigate to an internal page, submit a form and inspect the result. Very slow. The second problem is that they're very brittle, because a small enhancement to the application or a small change in the user-interface can easily break these tests.