**How does unit test**

1. Find the method that needs testing – should be simple and isolated, a unit test needs to test a single piece of code in isolation from the rest of the system.

2. Write the names of the tests that are required for the unit of code to get a feeling of what is required. A name should include the name of the method being tested and the purpose of the test, eg:

UpdatePasengerCount\_IncreasesCountByPassedAmount

CreateUserAccount\_ThrowsExceptionWhenEmailAddressIsNull

To find these names, read through the method. Ideally you want a test methods to exercise each side of if conditions, that will check for loops don’t miss any elements of a collection, exceptions are under the right circumstances, return dtos are populated with the correct values, calls to other services are made with the right parameter etc.

3. Start writing the simplest method by writing the assert clause which asserts the outcome of the method – this could be that value returned from the method under test matches an expected value or that the method under test calls another, mocked, service with the correct values.

4. Set up the data for the call to the method under test to give the desired result – This can be anything from creating dto parameters the test method requires to setting up returns from called services

5. Use mocks for external services that the method being tested uses – The unit of code being tested should be isolated from the rest of the system so when a test fails, it is easy to find where the problem is coming from. There are many mocking frameworks available which allow you so set up mocks, usually of interfaces, and define return values for methods on the interface. This allows you to control the execution of the method being tested.

6. Add the call to the method in the unit test and run the test– If everything has been set up correctly and the method being tested passes, the test should pass. If not, track down why the test is failing, is there a bug in the mehtod being tested or something missing from the test data setup.

7. Write the remaining tests for the method under test using the same steps – As you find common pieces of setup code, refactor them to separate methods multiple unit tests can use.

8. Once all the tests around a unit of code have been written, refactor the unit of code. Any problems introduced by the refactor will be caught by your unit tests and reported as failures.