**What is a Unit Test?**

Unit Testing involves testing independent units of code (such as functions) to ensure that the units of code and in turn the application as whole work as expected.

While it is not necessary to rely on additional frameworks or libraries to carry out the unit tests, using a library/ framework provides the following benefits:

1) Automated testing: Testing frameworks can be used to run a series of unit tests during the build (compilation, minification and packaging) phase of the code. Successful completion of unit tests will result in the generation of the application distributable artefact.

2) Simulate mock environments: For scenarios involving fetching data from a server via an ajax call or accessing a database, the testing framework allows mocking the external dependencies, so that we can focus on testing the individual unit of work that relies on these external dependencies.

3) Provide error details: Unit testing frameworks provide details of what went wrong during the unit tests. This helps developers to identify and fix the defects.

4) Facilitate code coverage: Unit testing frameworks facilitate the generation of test reports that detail the test coverage in the application. The test coverage gives an idea of the extent to which the code is tested and thereby increases our confidence in the system.

**Approaches to Unit Testing**

**Test Driven Development**

1. Start by writing a test
2. Run the test and any other tests. At this point, your newly added test should fail. If it doesn’t fail here, it might not be testing the right thing and thus has a bug in it.
3. Write the minimum amount of code required to make the test pass
4. Run the tests to check the new test passes
5. Optionally refactor your code
6. Repeat from 1

Test Driven Development emphasises on writing the unit tests before writing the implementation. A common grouse against TDD is:

1) Unit Tests must be written before the actual implementation. A lot of times, developers may not have a clear understanding of the requirements wrt testing. Moreover, delivery deadlines and client priorities, force them to start the implementation without thinking through the testing details.

2) TDD tests the implementation. Any change in the actual implementation, will result in revisiting the unit tests and making the necessary changes

**Behavior Driven Development**

Unlike TDD which focuses on the implementation details in a code, BDD is much more focused on the intended behaviour rather than how it is implemented. This gives developers the benefit of focusing on meeting the intended requirements of the application without worrying about rewriting tests when the implementation changes. While BDD can be used along with TDD, BDD approach can be used even after a considerable code has been written.

**What are the Javascript Unit Test Frameworks Components?**

1) Test Runner: Responsible for a spawning a simulated server environment that is used for testing

2) Test Framework: Responsible for setting up and tearing down of the test suite. Facilitates generation of test coverage reports and provides intelligible messages during test case failure.

3) Assertion Library: Responsible for testing / asserting /verifying that the conditions specified in the units are met by the implementation code

What are the Javascript Unit Test Tools ?

Javascript provides a number of tools for unit testing.

1) Jasmine: Contains all the Unit Test components: Test Runner, Test Framework, Assertion Library

2) Karma + Mocha + Chai : Karma as the Test Runner, Mocha as the Test Framework and Chai as the Assertion Library.

Few observations:

1) Jasmine seems like a very convenient library for Javascript unit tests.

i) Jasmine is more suitable for front end Javascript

ii) The Jasmine development community is not as active as Karma / Mocha

2) Karma + Mocha + Chai can be used to test front-end and server-side Javascript testing

i) Since Karma, Mocha and Chai are not tightly coupled, any of these testing components can be replaced easily.

ii) VueJS framework which is based on Webpack build and dependency management system, integrates well with Karma. Infact, VueJS by default recommends using Karma+Mocha+Chai for testing.

iii) Karma/ Mocha / Chai has an active development community and good documentation.

We have decided to try Karma/ Mocha / Chai as the Javascript Unit Test tools. The experience of writing preliminary tests using these tools will allow us to validate this decision.

**References:**

1) <https://simpleprogrammer.com/2010/10/15/the-purpose-of-unit-testing/>

2) <https://codeutopia.net/blog/2015/03/01/unit-testing-tdd-and-bdd/>

3) <https://vuejs.org/v2/guide/unit-testing.html>

4) <https://medium.com/@jaysoifer/how-to-test-vue-js-applications-c9bc3749adbb>

5) <https://medium.com/powtoon-engineering/a-complete-guide-to-testing-javascript-in-2017-a217b4cd5a2a>