# Test Driven Development Summary

## Roadmap for writing tests using mocha (JavaScript)

Whenever you start changing/adding to JavaScript that has tests related to it, make sure to be in the client folder and run npm test in a terminal window. This will alert you if any changes you make already existing tests fail! This means you can fix them right there, rather than have the problem go unnoticed for a while further down the repository.

Tests should be grouped by file & function, using describe statements.

If tests for a unit already exist, feel free to just add another it() to the describe block.

Types of tests:

* Defined tests: is the function/ any return value defined? (can help stop “not-quite obsolete” code being removed
* Type checking on return types: stop unexpected behaviour in other areas of the program where wrong types are returned
* Expected value tests: For each function that return a value, write a few tests with expected in -> out values (also consider edge cases here!)
* Other: for e.g. the sorting, database logic you will probably have to use some other assertions than equals. See the chai documentation for more information: <https://www.chaijs.com/api/assert/>

## Test Driven Development video:

Link: <https://youtu.be/9keJ61M9xlg>

In the video TDD was applied to write a few functions used to calculate the distance between two locations (latitude, longitude) on Earth.

The slightly simplified (from user stories) acceptance test for this functionality was does calculating the distance from x to y give a comparable value when these values are put into an online distance calculator.

After the video, TDD continued on the sorting logic after some refactoring of the metaInfo data structure. This ultimately ended up passing the user story acceptance test of having the list sorted from low to high.

## Evidence of tests being used:

GitHub issues/ pull requests tagged with TDD:

<https://github.com/raufnawaztarar/Team-16-agile/issues?utf8=✓&q=label%3Atdd>

We had Travis setup during part of Sprint 2, this meant that our JavaScript Unit tests were ran every time somebody pushed or opened a pull request! This helped stop failing tests going unnoticed.

In the end our test coverage ended up being “fairly limited”. There were rarely situations where tests failed other than while writing new ones. I believe a higher test coverage would have been beneficial to our project.