Topic 16: Validation

CITS3403 Agile Web Development

Getting MEAN with Mongo, Express, Angular and Node, Chapter 11

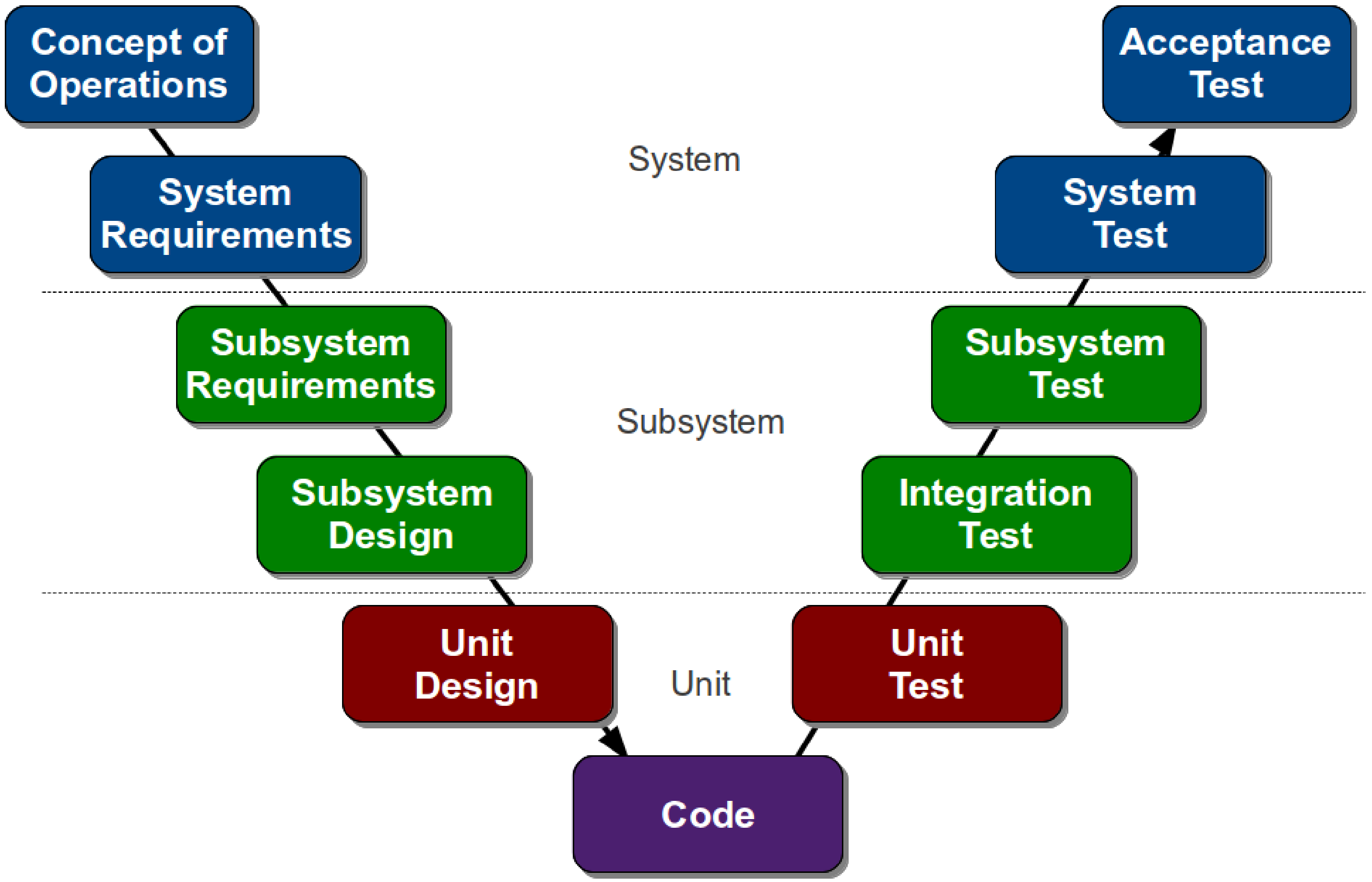
Semester 1, 2018

Verification and Validation

- Writing a bug free application is critical to to success of that application.
- There are various ways to eliminate bugs.
  - Code inspections: having peers critically examine your code and make suggestions.
  - Formal verification: building precise specifications of correctness, and proving the code meets these specs.
  - Testing: Providing test cases of inputs and actions, and expected behaviors.
- Testing is the most common and what we will focus on here.

The V-model

- The V-model links types of tests to stages in the development process.



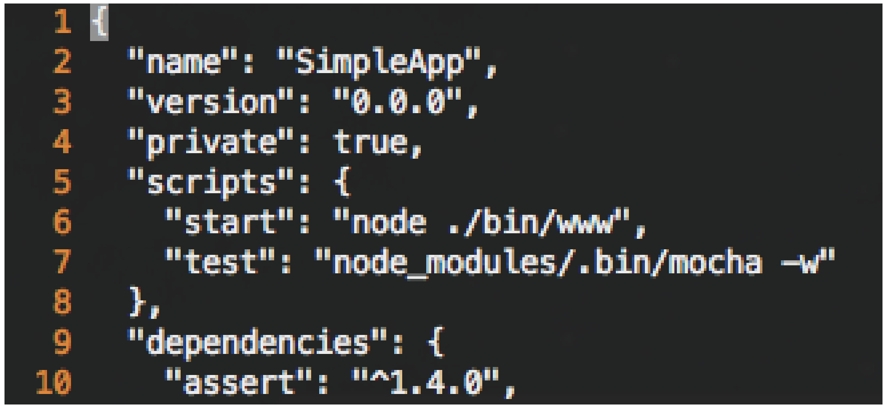
Types of test

- Unit Tests: test each individual function for to ensure it behaves correctly (2-5 tests per function)
- Integration Test: Execute each scenario to make sure modules integrate correctly.
- System Test: Integrate real hardware platforms.
- Acceptance Test: Run through complete user scenarios via the user interface.

Aim to catch most bugs with unit and integration tests, and focus on automating these.

Incorporating tests into Express

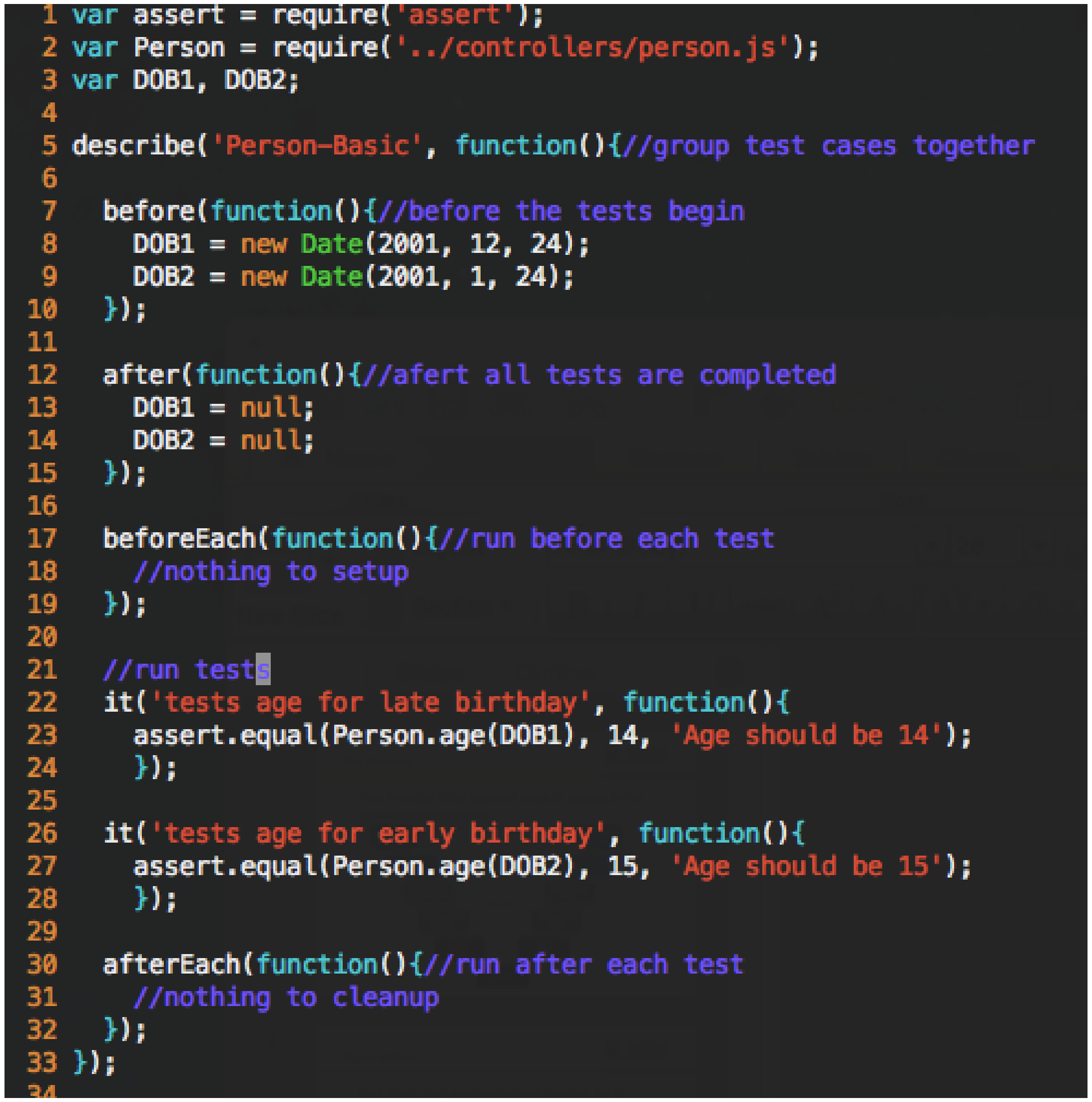
- We’ll use Mocha as a test framework for our Express project. To set up:
- `npm install --save mocha`
- `npm install --save assert`
- Assert is the assertion library (there are many).
- Create a directory for the tests in the project root:
- `mkdir test`
- Setup tests in the `package.json` file.
- test with `npm test`



```
{
  "name": "SimpleApp",
  "version": "0.0.0",
  "private": true,
  "scripts": {
    "start": "node ./bin/www",
    "test": "node_modules/.bin/mocha -w"
  },
  "dependencies": {
    "assert": "^1.4.0",
```

Anatomy of a test:

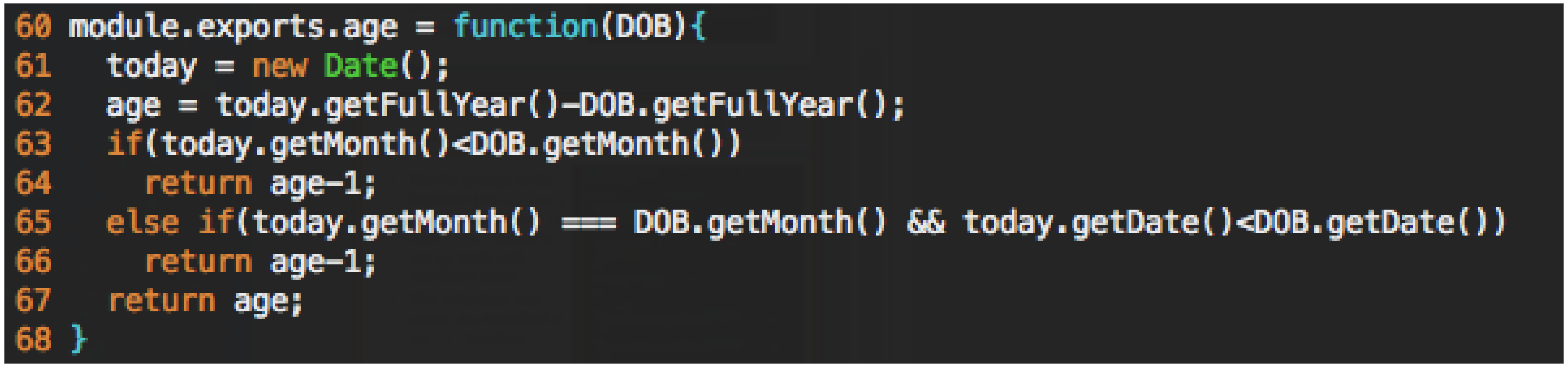
- Mocha groups tests together (`describe`), and provides `before` and `after` hooks to set up tests and teardown tests.
- The individual test cases are specified in an `it` function.



```
var assert = require('assert');
var Person = require('../controllers/person.js');
var DOB1, DOB2;

describe('Person-Basic', function(){//group test cases together

  before(function(){//before the tests begin
    DOB1 = new Date(2001, 12, 24);
    DOB2 = new Date(2001, 1, 24);
  });

  after(function(){//afert all tests are completed
    DOB1 = null;
    DOB2 = null;
  });

  beforeEach(function(){//run before each test
    //nothing to setup
  });

  //run tests
  it('tests age for late birthday', function(){
    assert.equal(Person.age(DOB1), 14, 'Age should be 14');
    });

  it('tests age for early birthday', function(){
    assert.equal(Person.age(DOB2), 15, 'Age should be 15');
    });

  afterEach(function(){//run after each test
    //nothing to cleanup
  });
});
```

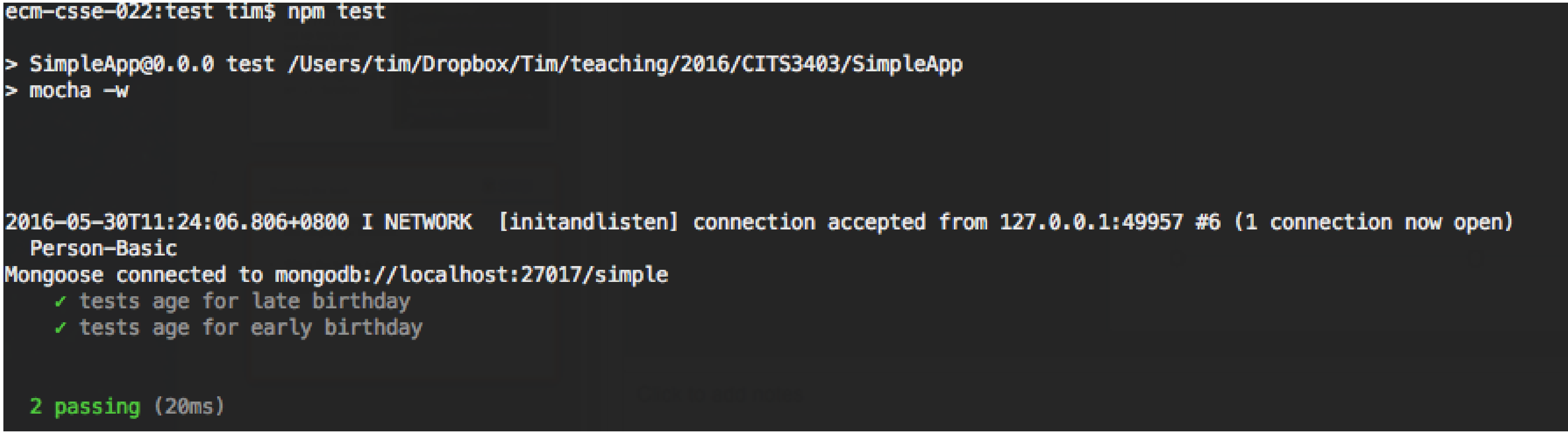
Running the test:

- The test is designed for the simple age function.



```javascript
module.exports.age = function(DOB){
  today = new Date();
  age = today.getFullYear()-DOB.getFullYear();
  if(today.getMonth()<DOB.getMonth())
    return age-1;
  else if(today.getMonth() === DOB.getMonth() && today.getDate()<DOB.getDate())
    return age-1;
  return age;
}
```

- When the test is run, Mocha will provide a report:



```
ecm-csse-022:test tim$ npm test

> SimpleApp@0.0.0 test /Users/tim/Dropbox/Tim/teaching/2016/CITS3403/SimpleApp
> mocha -w




2016-05-30T11:24:06.806+0800 I NETWORK  [initandlisten] connection accepted from 127.0.0.1:49957 #6 (1 connection now open)
  Person-Basic
Mongoose connected to mongodb://localhost:27017/simple
    ✓ tests age for late birthday
    ✓ tests age for early birthday


  2 passing (20ms)
```

Assertion libraries:

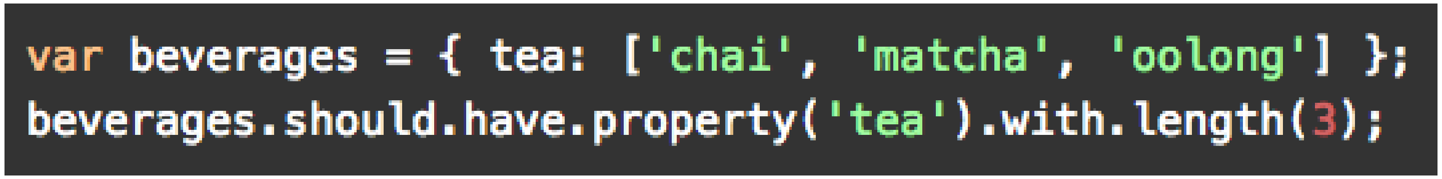
- There are many assertion libraries: assert is one of the most basic.
- assert states a property that should hold, and throws an assertion error if it doesn’t (the test fails).
- The possible operators are:

```
assert(value[, message])
assert.deepEqual(actual,
                 expected[, message])
assert.deepStrictEqual(actual,
                 expected[, message])
assert.doesNotThrow(block[, error]
                 [, message])
assert.equal(actual,
                 expected[, message])
assert.fail(actual, expected,
                 message, operator)
assert.ifError(value)
```

```
assert.notDeepEqual(actual,
                 expected[, message])
assert.notDeepStrictEqual(actual,
                 expected[, message])
assert.notEqual(actual,
                 expected[, message])
assert.notStrictEqual(actual,
                 expected[, message])
assert.ok(value[, message])
assert.strictEqual(actual,
                 expected[, message])
assert.throws(block[, error]
                 [, message])
```

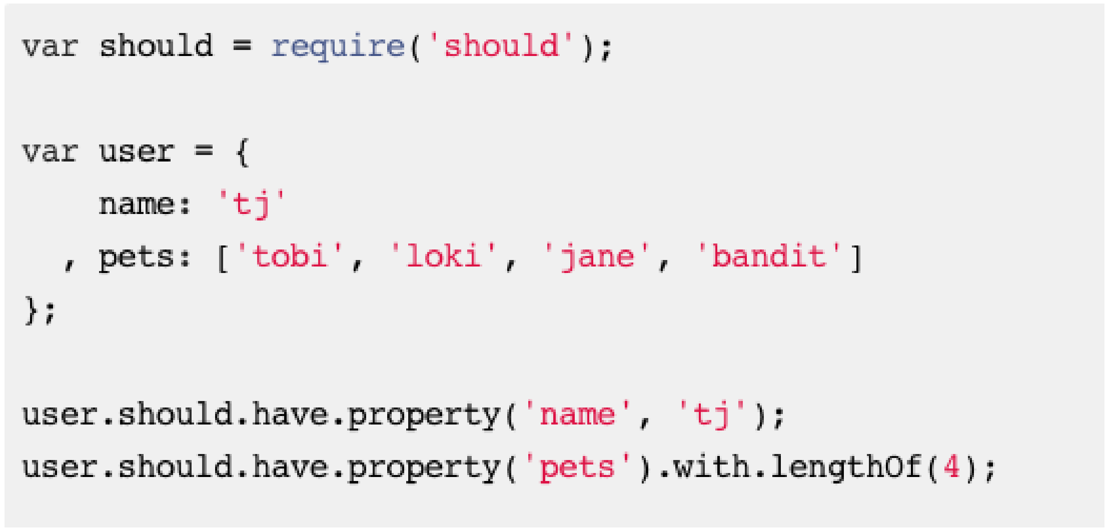
Variations

- There are different assertion libraries to suit different styles of testing.
- `chai` uses *behavior driven development style* :



```
var beverages = { tea: ['chai', 'matcha', 'oolong'] };
beverages.should.have.property('tea').with.length(3);
```

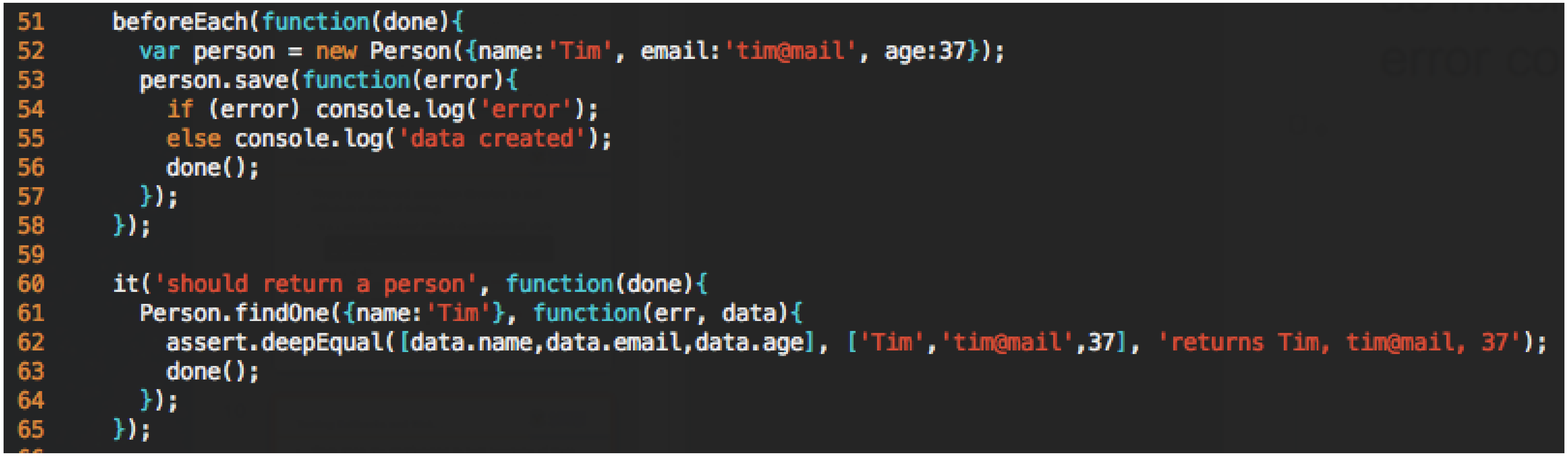
- `should` is also a popular alternative with a similar style:



```
var should = require('should');

var user = {
    name: 'tj'
  , pets: ['tobi', 'loki', 'jane', 'bandit']
};

user.should.have.property('name', 'tj');
user.should.have.property('pets').with.lengthOf(4);
```

Testing Callbacks and Web…

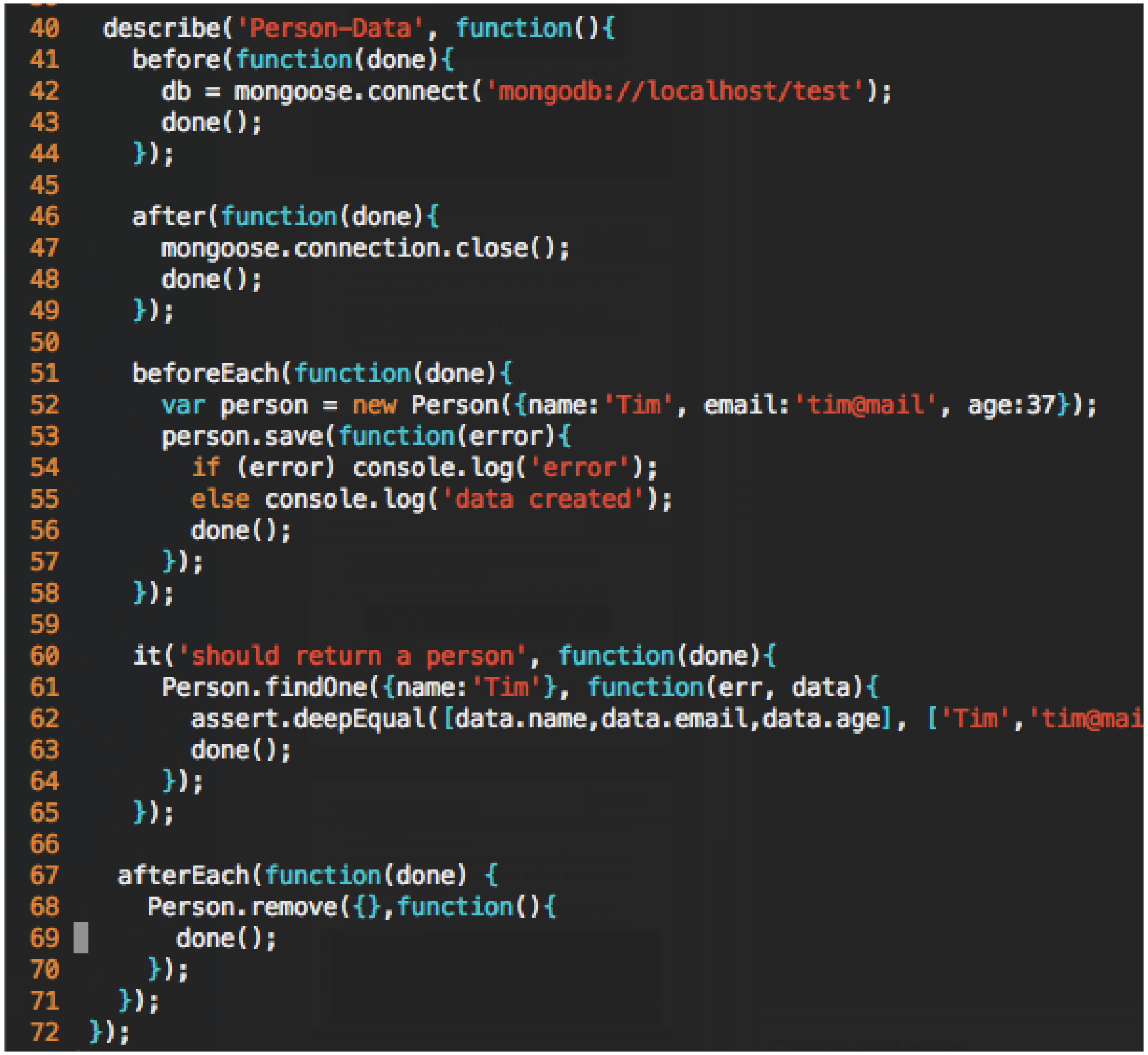
- Web apps are challenging to test because of their asynchronous nature.
- A function can complete before its callbacks have, so mocha may report success before an assertion error could occur.
- Mocha provides a callback function done() that forces the test to wait until all call backs are done.



```
beforeEach(function(done){
  var person = new Person({name:'Tim', email:'tim@mail', age:37});
  person.save(function(error){
    if (error) console.log('error');
    else console.log('data created');
    done();
  });
});

it('should return a person', function(done){
  Person.findOne({name:'Tim'}, function(err, data){
    assert.deepEqual([data.name,data.email,data.age], ['Tim','tim@mail',37], 'returns Tim, tim@mail, 37');
    done();
  });
});
```

Working with Mongo:

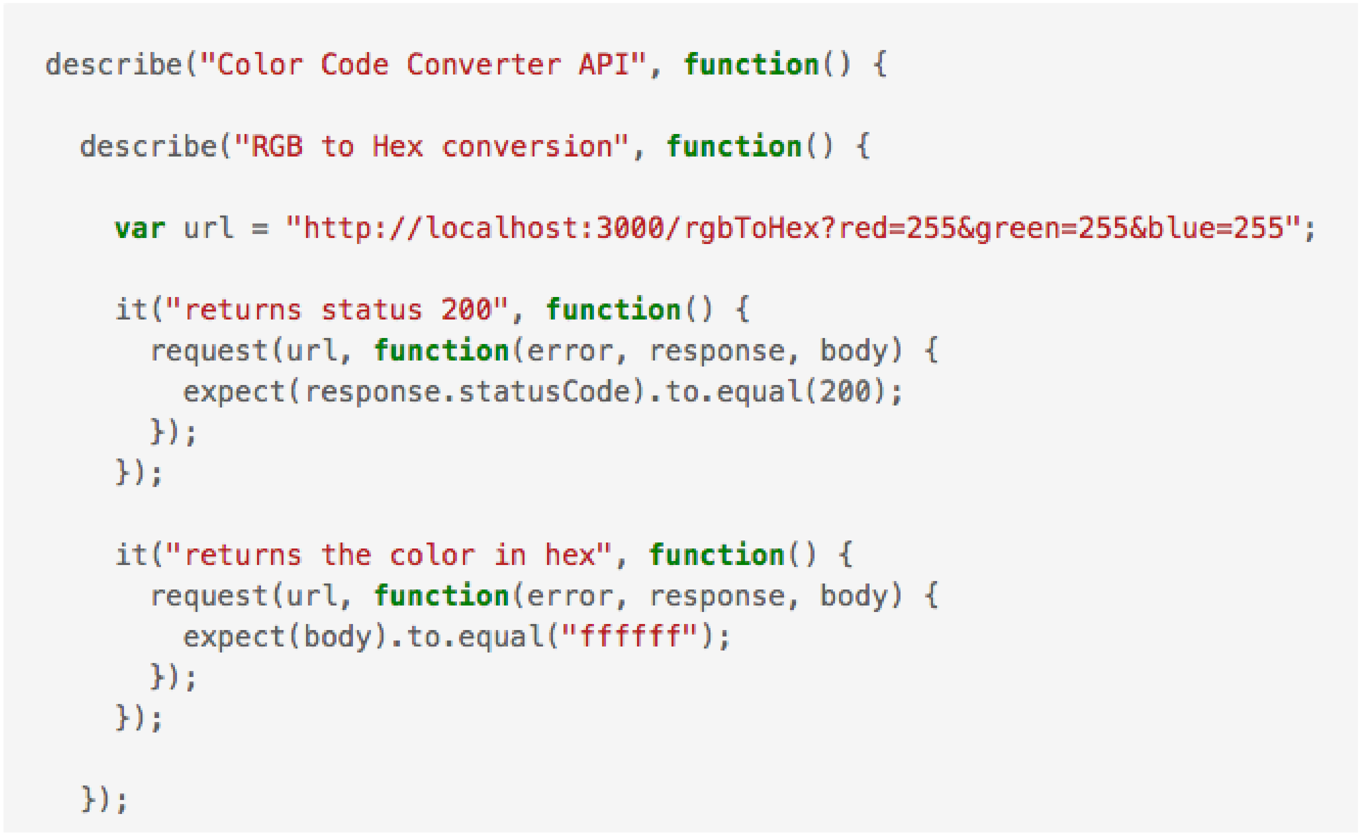
- The database connections should be opened and closed at the start and end of the test.
- Note we use deepEqual here because we are comparing arrays.



```
 describe('Person-Data', function(){
   before(function(done){
     db = mongoose.connect('mongodb://localhost/test');
     done();
   });

   after(function(done){
     mongoose.connection.close();
     done();
   });

   beforeEach(function(done){
     var person = new Person({name:'Tim', email:'tim@mail', age:37});
     person.save(function(error){
       if (error) console.log('error');
       else console.log('data created');
       done();
     });
   });

   it('should return a person', function(done){
     Person.findOne({name:'Tim'}, function(err, data){
       assert.deepEqual([data.name,data.email,data.age], ['Tim','tim@mai
       done();
     });
   });

  afterEach(function(done) {
   Person.remove({},function(){
    done();
   });
  });
});
```

Working with web requests.

- The npm request package can be used to test that routes are giving correct responses in integration tests



```
describe("Color Code Converter API", function() {

  describe("RGB to Hex conversion", function() {

    var url = "http://localhost:3000/rgbToHex?red=255&green=255&blue=255";

    it("returns status 200", function() {
      request(url, function(error, response, body) {
        expect(response.statusCode).to.equal(200);
      });
    });

    it("returns the color in hex", function() {
      request(url, function(error, response, body) {
        expect(body).to.equal("ffffff");
      });
    });

  });
```

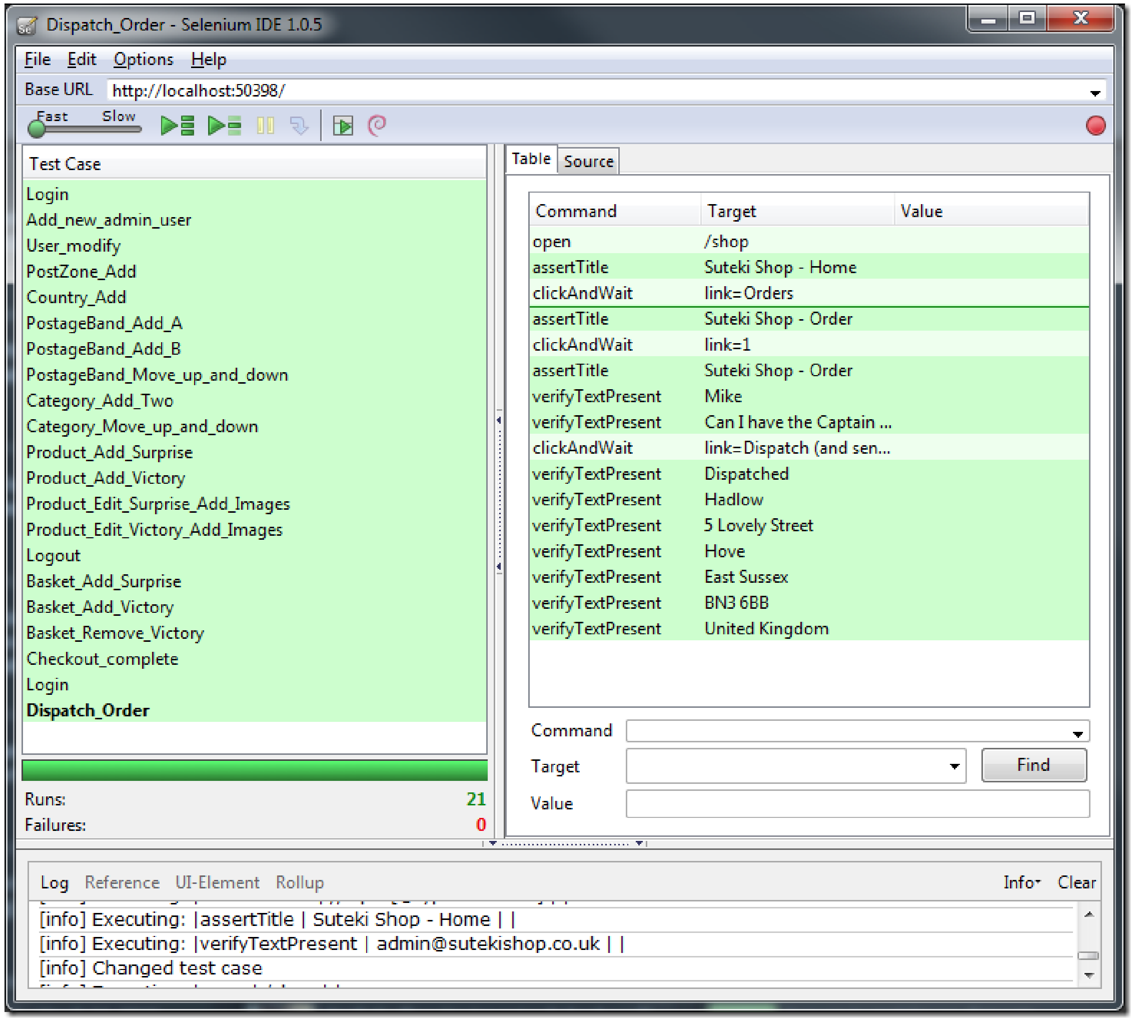
https://semaphoreci.com/community/tutorials/getting-started-with-node-js-and-mocha

Using tests in development

- Testing is an integral part of development.
- You should aim for 100% test coverage. Every line of code should execute in at least one test.
- Test driven development is a process where you write tests first, and then write code *just* to pass the tests.
- Tests can be integrated into the build environment in continuous integration: *Travis CI* or *Drone* can be configured so that every time you push an update, the code is automatically tested and launched (if it passes.

User Tests

- User testing is more challenging since it depends on the end user environment.
- Selenium can be used to automate browsers to run test cases.
- PhantomJS is a headless browser that can be used for testing with Mocha


Dispatch_Order - Selenium IDE 1.0.5
File Edit Options Help
Base URL http://localhost:50398/
Fast Slow
Test Case
Table Source
Command
Target
Value
Find
Log Reference UI-Element Rollup
Info Clear

Test Case
Login
Add_new_admin_user
User_modify
PostZone_Add
Country_Add
PostageBand_Add_A
PostageBand_Add_B
PostageBand_Move_up_and_down
Category_Add_Two
Category_Move_up_and_down
Product_Add_Surprise
Product_Add_Victory
Product_Edit_Surprise_Add_Images
Product_Edit_Victory_Add_Images
Logout
Basket_Add_Surprise
Basket_Add_Victory
Basket_Remove_Victory
Checkout_complete
Login
Dispatch_Order

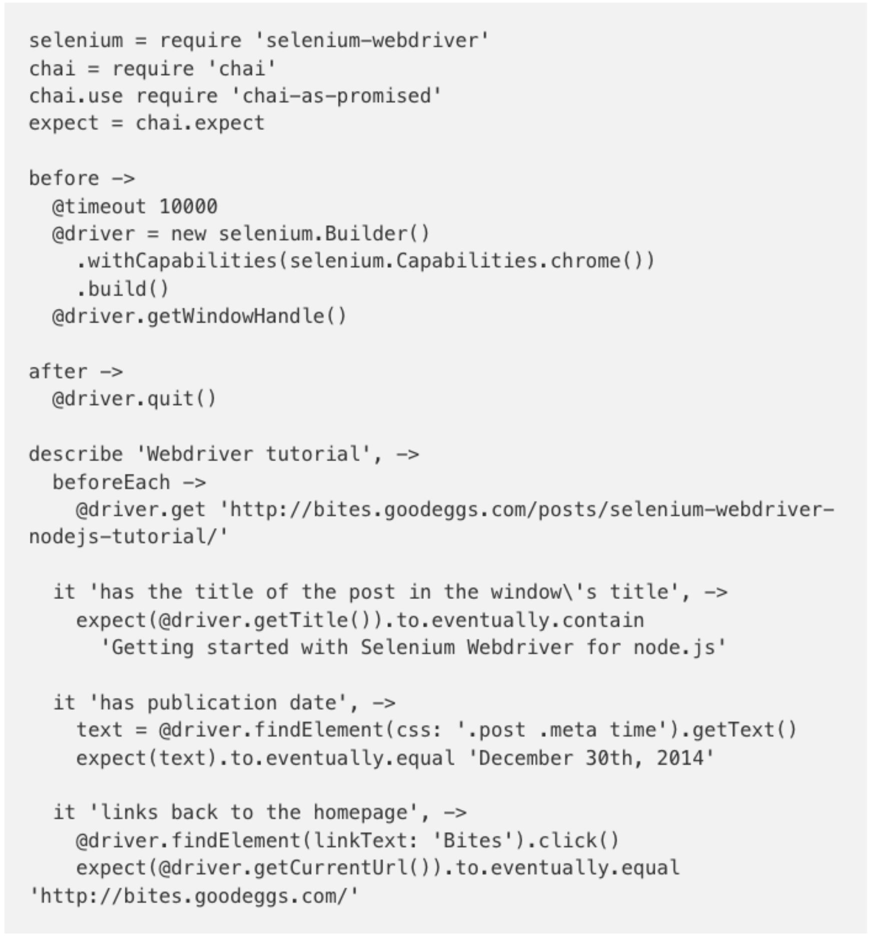
Runs: 21
Failures: 0

Command	Target	Value
open	/shop	
assertTitle	Suteki Shop - Home	
clickAndWait	link=Orders	
assertTitle	Suteki Shop - Order	
clickAndWait	link=1	
assertTitle	Suteki Shop - Order	
verifyTextPresent	Mike	
verifyTextPresent	Can I have the Captain ...	
clickAndWait	link=Dispatch (and sen...	
verifyTextPresent	Dispatched	
verifyTextPresent	Hadlow	
verifyTextPresent	5 Lovely Street	
verifyTextPresent	Hove	
verifyTextPresent	East Sussex	
verifyTextPresent	BN3 6BB	
verifyTextPresent	United Kingdom	

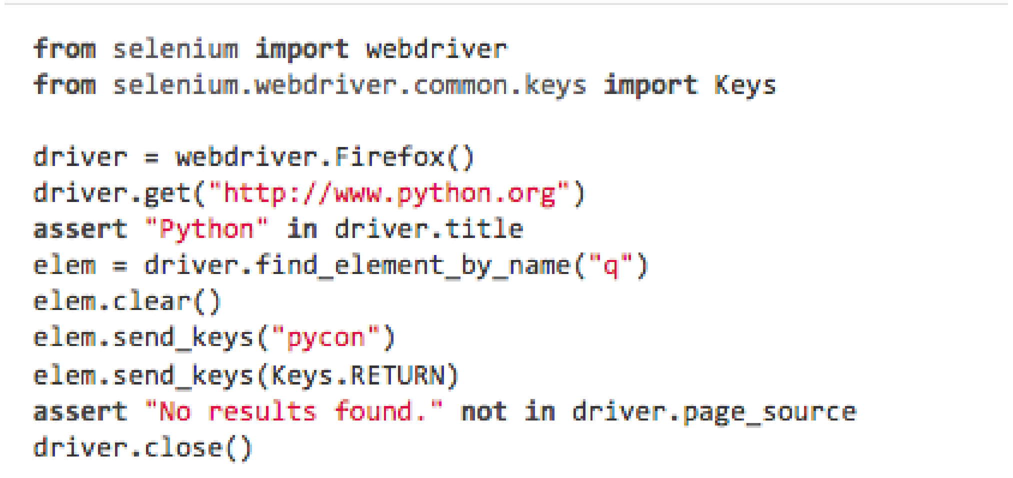
```
[info] Executing: |assertTitle | Suteki Shop - Home | |
[info] Executing: |verifyTextPresent | admin@sutekishop.co.uk | |
[info] Changed test case
```

Selenium

- A tool set that automates web app testing across platforms
- Can simulate user interactions in browser
- Two components
  - Selenium IDE
  - Selenium WebDriver (aka. Selenium 2)



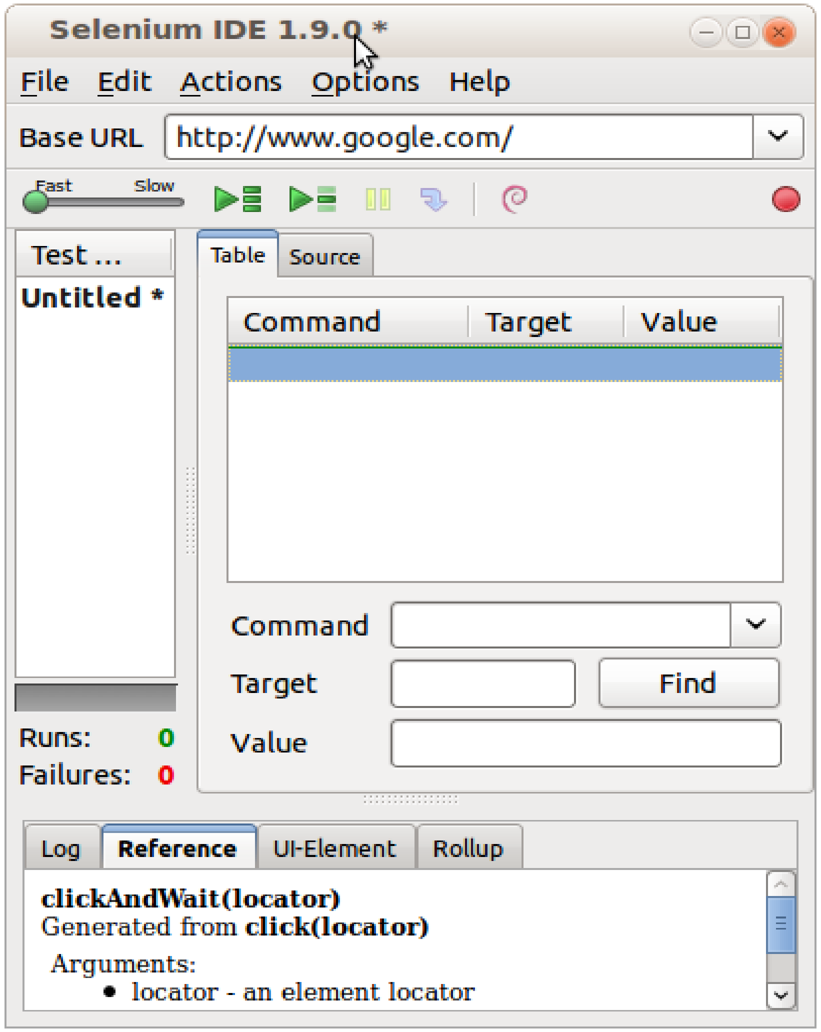
```
selenium = require 'selenium-webdriver'
chai = require 'chai'
chai.use require 'chai-as-promised'
expect = chai.expect

before ->
  @timeout 10000
  @driver = new selenium.Builder()
    .withCapabilities(selenium.Capabilities.chrome())
    .build()
  @driver.getWindowHandle()

after ->
  @driver.quit()

describe 'Webdriver tutorial', ->
  beforeEach ->
    @driver.get 'http://bites.goodeggs.com/posts/selenium-webdriver-
nodejs-tutorial/'

  it 'has the title of the post in the window\'s title', ->
    expect(@driver.getTitle()).to.eventually.contain
      'Getting started with Selenium Webdriver for node.js'

  it 'has publication date', ->
    text = @driver.findElement(css: '.post .meta time').getText()
    expect(text).to.eventually.equal 'December 30th, 2014'

  it 'links back to the homepage', ->
    @driver.findElement(linkText: 'Bites').click()
    expect(@driver.getCurrentUrl()).to.eventually.equal
'http://bites.goodeggs.com/'
```



```
from selenium import webdriver
from selenium.webdriver.common.keys import Keys

driver = webdriver.Firefox()
driver.get("http://www.python.org")
assert "Python" in driver.title
elem = driver.find_element_by_name("q")
elem.clear()
elem.send_keys("pycon")
elem.send_keys(Keys.RETURN)
assert "No results found." not in driver.page_source
driver.close()
```

Selenium IDE

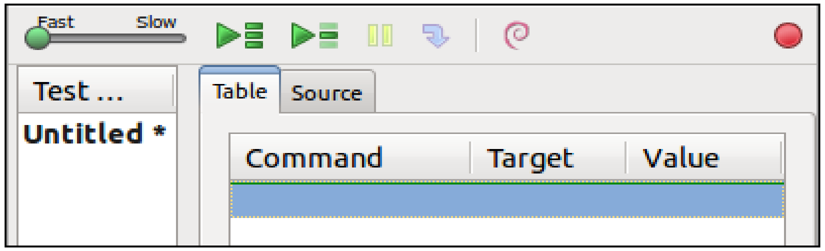
- Firefox extension
- Easy record and replay
- Debug and set breakpoints
- Save tests in HTML, WebDriver and other formats.



clickAndWait(locator)
Generated from **click(locator)**
Arguments:
- locator - an element locator

Selenium IDE test cases

- Selenium saves all information in an HTML table format



- Each record consists of:
  - **Command** – tells Selenium what to do (e.g. "open", "type", "click", "verifyText")
  - **Target** – tells Selenium which HTML element a command refers to (e.g. textbox, header, table)
  - **Value** – used for any command that might need a value of some kind (e.g. type something into a textbox)

How to record/replay with Selenium IDE

1. Start recording in Selenium IDE
2. Execute scenario on running web application
3. Stop recording in Selenium IDE
4. Verify / Add assertions
5. Replay the test.

... or using webdriver you can integrate selenium with any unit testing scripting language.

You can test functionality, responsiveness and general usability.