Testing Asynchronous Code

What is Asynchronous Code?

- Asynchronous calls return immediately and continue to run in the background.
- Asynchronous calls generally notify the caller that they have completed their work either via a callback function, a Javascript “promise”, or the new async/await javascript keywords.
- Examples of asynchronous calls are:
  - Timers (i.e. setTimeout)
  - HTTP Requests (i.e. http.get)
  - Database Operations

Async Testing of Callbacks

```
function myAsyncFunction(callback){
  setTimeout(function(){
    callback(“blah”);
  }, 50);
}

it(‘callback test’, function(done){
  myAsyncFunction(function(str){
    expect(str).to.equal(“blah”);
    done();
  });
});
```

- To test asynchronous code with callbacks pass a “done” parameter to your test.
- This is a callback function provided by Mocha.
- Mocha will not complete the test until the “done” callback has been called.

Async Testing with Promises

```
function promiseFunc(){
 return new Promise(
   (resolve, reject)=>{
     setTimeout(()=>{
      resolve(“blah”);}, 50);
    });
}
it(“promise test”, function(){
  return promiseFunc().then(res=>{
     expect(res).to.equal(“blah”);
  });
});
```

- To test asynchronous code with promises you simply return the promise from your test.
- Mocha delays the test until the returned promise is resolved.

Async Testing with Async/Await

```
function promiseFunc(){
 return new Promise(
   (resolve, reject)=>{
     setTimeout(()=>{
      resolve(“blah”);}, 50);
    });
}
it(“await test”, async ()=>{
  var res = await promiseFunc();
  expect(res).to.equal(“blah”);
});
```

- To test with the async/await keywords specify “async” on your unit test.
- Inside your test you then call “await” on the asynchronous function that you’re testing.
- Your unit test will return a promise which Mocha will wait to be resolved.