Testing Express Routes

Agenda

1. Learning Objectives (5 minutes)
2. Overview: Testing Controllers (20 minutes)
3. BREAK (10 minutes)
4. Activity: Write Route Tests for Messages API (50 minutes)
5. Wrap-Up

Learning Objectives

By the end of this lesson, you should be able to…

1. Write unit tests for routes using the `chaiHttp` library.
2. Use `describe()` blocks to logically arrange unit tests.
3. Use the `beforeEach()` and `afterEach()` functions to specify test set-up and tear-down routines.

Warm-Up: Write a Mocha Test (5 minutes)

How would we write a test for the following function?

```
const celsiusToFahrenheit = (celsius) => {
    return celsius * 9 / 5 + 32
}
```

Use your `tdd-bdd-challenge` solutions as a guide.

Overview: Testing Controllers (20 minutes)

Using `chaiHttp` to test our routes

We can test our controllers using the `chaiHttp` library.

Here is some example code for a “Hello, World” route:

```
/* src/index.js */

const express = require('express');
const app = express();
const PORT = env.process.PORT || 8080;

app.get('/', (req, res) => {
    res.send({
        message: 'Hello, world!'
    });
});
```

```
app.listen(port, () => {
    console.log(`Listening on localhost:${port}`);
})

module.exports = app;
```

And here is some example test code:

```
/* src/test/index.js */

const chai = require('chai');
const expect = chai.expect;
const chaiHttp = require('chai-http');
const app = require('../app.js');

chai.use(chaiHttp);

// Describe is like a container for our tests.
describe('Hello World Route Test', () => {

    // Test Case 1
    it('Returns a 200 Response', (done) => {
        chai.request(app)
            .get('/')
            .end((error, response) => {
                if (error) done(error);
                expect(response).to.have.status(200);
                done();
            });
    });

    // Test Case 2
    it('Returns a "Hello World" message', (done) => {
        chai.request(app)
            .get('/')
            .end((error, response) => {
                if (error) done(error);
                expect(response.body).to.be.deep.equal({
                    message: 'Hello, world!'
                });
                done();
            });
    });
});
```

Introducing: `beforeEach` and `afterEach`

Sometimes, there are things you *always* want to do either **before** each test runs (think of these like setup), or **after** each test ru
the `beforeEach()` and `afterEach()` functions.

Here is an example:

```
// Create a sample user for use in tests.
beforeEach((done) => {
```

```
    const sampleUser = new User({
        username: 'myuser',
        password: 'mypassword',
        _id: SAMPLE_OBJECT_ID
    })
    sampleUser.save()
    .then(() => {
        done()
    })
})

// Delete sample user.
afterEach((done) => {
    User.deleteMany({ username: ['myuser', 'anotheruser'] })
    .then(() => {
        done()
    })
})
```

BREAK (10 minutes)

Activity: Write Route Tests for Messages API (30 minutes)

Go to the **starter code** for the Messages API. The routes and unit tests should already be completed for the `/user` endpoints.

You will be adding unit tests for the `/messages` endpoints. Try writing these using the TDD methodology:

1. Write a test
2. Run the test (it should fail)
3. Write code for the route being tested
4. Run the test again. If it passes, great! If not, revise your route or test code until it passes.
5. Repeat for the next test!

Wrap-Up

Complete the **Chai Testing Challenges** as homework before our next class.