CSCE 120: Learning To Code

Introduction To Data
Hacktivity 2.2

Introduction

Prior to engaging in this hacktivity, you should have completed all of the pre-class activities as outlined in this module. At the start of class, you will be randomly assigned a partner to work with on the entirety of this hacktivity as a *peer programming* activity. Your instructor will inform you of your partner for this class.

One of you will be the driver and the other will take on the navigator role. Recall that a driver is in charge of the keyboard and computer while the navigator is in charge of the handout and directing the activity. However, you are *both* responsible for contributing and discussing solutions. If you were a driver/navigator in the prior activity, switch roles for this activity.

Download Your Starter Code

If you are using a different computer than for the previous Hacktivity, you may need to download the project code. Point your browser to https://github.com/cbourke/RosterApp and download the project code for this hacktivity. Unzip the file to your CSCE 120 folder and drag it into your Light Table workspace.

1 Creating Data

One essential component to software development is testing. There are many different types of testing (unit testing, integration testing, load testing, etc.). Testing provides a level of assurance that the software developed is free of bugs and will work as intended.

One essential component to testing is test data. Designing proper test data involves almost as much work and skill as developing the software itself. *Test Driven Development* (TDD) software engineering techniques have refocused attention to developing proper test cases, test data and testing techniques.

1.1 Manually Creating Test Data

In this exercise, you'll develop some test data.

1. Backup the `roster.json` data file (copy it to `roster.old.json`) and clear its contents.
2. Create your own test data for the roster table application. Include you and your partner's course data for this semester. Alternatively, you can create dummy data if you do not want to use personal data.
3. Validate your test data using JSON Lint and reevaluate the `roster.html` page to verify it is correct.
4. Fix any problems or errors until the page loads correctly.

As with data transformation, creating your own test data is tedious and error prone. Manually creating a small amount of data may not be sufficient to rigorously test your code. Moreover, if a test case fails, how do you know that the problem lies with the code or the test case? If the test case is in error, this is referred to as a false-positive. What if both are in error: a faulty test case could hide a bug. This is referred to as a false-negative.

1.2 Generating Test Data

In the next activity, we'll use a tool to generate test cases for us.

1. Point your browser to http://www.generatedata.com/ (alternatively, http://www.json-generator.com/)
2. Use this online tool to generate a large number (say 100) test cases conforming to the expected format of the roster data. Note that you may not be able to get meaningful course data, but choose something close.
3. Cut and paste the generated data into your JSON file and run the application. Fix any errors.

2 Data Organization & Integrating a Dynamic Plugin

The application we've been using displays roster data in a plain table. Unfortunately, it is static (its size and ordering are fixed and cannot be changed), not easily readable (columns and rows are not delineated), any overall data (such as number of seniors, etc.) are not easily discernible, etc.

In this activity, we'll integrate a jQuery plugin called DataTables (see http://www.datatables.net/).

1. Copy-paste the following lines into your `roster.html` file in the `<head>` element after the jQuery line:

```
<script src="https://cdn.datatables.net/1.10.7/js/jquery.dataTables.min.js"></script>
<link rel="stylesheet" type="text/css" href="https://cdn.datatables.net/1.10.7/css/jquery.dataTables.min.css"/>
```

2. Open the `roster.js` JavaScript source file and add the following line below the `//TODO: initialize data table` line:

```
$("#roster").dataTable();
```

3. Reload your page and view the results.
4. Play with the features that this plugin offers and discuss with your partner what a user could now do with this data that they couldn't before.

This demonstrates that it doesn't take a lot to get dynamic, sleek looking elements into a webpage when you leverage the power of libraries and plugins.

3 Load Testing

Load testing is a software engineering process of placing a high demand or "load" on a program or system to see how well it performs under these conditions. Load testing can reveal inefficiencies that should be addressed or the limits of a program.

In this activity you will do some basic load testing on the roster application to see how well it performs with more and more data. Unfortunately, the test generation services you used before will not allow you to generate more than 100 items. Instead we'll simulate lots of data by repeatedly adding duplicate rows to the table.

1. Open the `roster.js` file.
2. Navigate to the code that adds ("appends") each record to the table: `$("#roster tbody").append(item);`
3. Surround this line of code with a *loop* that repeats the add operation 10 times. The code should look like the following:

```
for(var j=0; j<10; j++) {
  $("#roster tbody").append(item);
}
```

4. Reload the page and observe the results. How long did it take to load the page? How long does it take to search/filter results?

5. Change the value that controls how many times each record is repeated. Increase it to 20, 40, 80, 160 (that is double it each time) and time how long it takes for each of these tests. Predict how long it would take if you were to set it to 500, 1,000, and 10,000 (that is, 1 million records). Discuss this with your partner.