1.1 - Exercise (Instructions): Bootstrap and JQuery

Objectives and Outcomes

In this exercise we learn about using Bootstrap's JS component methods together with JQuery and JavaScript to write JavaScript code to control the JS component. We will use the Carousel as an example of a component that can be controlled. At the end of this exercise you will be able to:

- Use Bootstrap's JS component methods together with JQuery and Javascript
- Use JS code to control the Bootstrap JS component

Adding the Carousel Control Buttons

- We will introduce two new buttons into the carousel component that we already included in the index.html page. To add the two buttons to the carousel, add the following code to the end of the carousel:

```
<div class="btn-group" id="carouselButton">
    <button class="btn btn-danger btn-sm" id="carousel-pause">
        <span class="fa fa-pause"></span>
    </button>
    <button class="btn btn-danger btn-sm" id="carousel-play">
        <span class="fa fa-play"></span>
    </button>
</div>
```

We are adding the two buttons inside a button group with the ID carouselButtons. The two buttons contain the pause and play glyphicons to indicate their corresponding actions.

Adding CSS Class for the Buttons

- Next, we add the following CSS class to styles.css file to position the buttons at the bottom-right corner of the carousel:

```
#carouselButton {
    right:0px;
```

```
    position: absolute;
    bottom: 0px;
    z-index: 1;
}
```

Adding JavaScript Code

- Finally we add the following JavaScript code to activate the buttons:

```
<script>
    $(document).ready(function(){
        $("#mycarousel").carousel( { interval: 2000 } );
        $("#carousel-pause").click(function(){
            $("#mycarousel").carousel('pause');
        });
        $("#carousel-play").click(function(){
            $("#mycarousel").carousel('cycle');
        });
    });
</script>
```

- Do a Git commit with the message "Bootstrap JQuery"

Conclusions

In this exercise we learnt about Bootstrap's JS component methods and how they can be used together with JQuery and JavaScript to control the behavior of a Bootstrap JS component.

1.2 - Exercise (Instructions): More Bootstrap and JQuery

Objectives and Outcomes

In this exercise we extend the previous exercise of controlling the carousel by using more JQuery and JavaScript to write JavaScript code to control the JS component. At the end of this exercise you will be able to:

- Use Bootstrap's JS component methods together with JQuery and Javascript
- Use JS code to control the Bootstrap JS component

Modifying the Carousel Control Buttons

- We will modify the carousel control buttons in the carousel component that we already included in the index.html page. Instead of two buttons, we will use a single button that will indicate if the carousel is currently cycling or paused. Furthermore we can use the button to toggle the carousel cycling behavior:

```
<button class="btn btn-danger btn-sm" id="carouselButton">

   <span id="carousel-button-icon" class="fa fa-pause"></span>

</button>
```

We are adding a single button inside a button group with the ID carouselButton. The buttons will show either as a pause or play button based on the current behavior of the carousel.

Modifying JavaScript Code

- Finally we modify the JavaScript code to control the behavior of the carousel and also show the appropriate button:

```
$("#carouselButton").click(function(){

  if ($("#carouselButton").children("span").hasClass('fa-pause')) {

     $("#mycarousel").carousel('pause');

     $("#carouselButton").children("span").removeClass('fa-pause');

     $("#carouselButton").children("span").addClass('fa-play');

  }

  else if ($("#carouselButton").children("span").hasClass('fa-play')){
```

```
            $("#mycarousel").carousel('cycle');

            $("#carouselButton").children("span").removeClass('fa-play');

            $("#carouselButton").children("span").addClass('fa-pause');

        }

    });
```

- Do a Git commit with the message "More Bootstrap JQuery".

Conclusions

In this exercise we learnt more about Bootstrap's JS component methods and how they can be used together with JQuery and JavaScript to control the behavior of a Bootstrap JS component.

1.3 - Bootstrap and JQuery: Additional Resources

PDFs of Presentations

1-Bootstrap-JQuery.pdfPDF File

Bootstrap Resources

- Bootstrap Carousel Methods

JQuery

- JQuery
- W3Schools JQuery

2.1 - Exercise (Instructions): Less

Objectives and Outcomes

In this exercise you will learn to write Less code and then automatically transform it into the corresponding CSS code. At the end of this exercise you will be able to:

- Write Less code using many of the features of Less
- Automatically convert the Less code into CSS

Adding Less Variables

- Open the *conFusion* project in a text editor of your choice. In the css folder, create a file named *styles.less*. We will add the Less code into this file.
- Add the following Less variables into the file:

```
@lt-gray: #ddd;

@background-dark: #512DA8;

@background-light: #9575CD;

@background-pale: #D1C4E9;



// Height variables

@carousel-item-height: 300px;
```

We have just added a few color and a height variable. We will make use of these variables while defining the classes.

Less Mixins

- Next we add a mixin into the file as follows:

```
.zero-margin (@pad-up-dn: 0px, @pad-left-right: 0px) {

    margin:0px auto;

    padding: @pad-up-dn @pad-left-right;

}
```

We will make use of this to define several row classes next.

- Using the variables and Mixin class that we defined earlier, add the following row classes to the file:

```
.row-header{
    .zero-margin();
}
.row-content {
    .zero-margin(50px,0px);
    border-bottom: 1px ridge;
    min-height:400px;
}
.footer{
    background-color: @background-pale;
    .zero-margin(20px, 0px);
}
.jumbotron {
    .zero-margin(70px,30px);
    background: @background-light ;
    color:floralwhite;
}
address{
    font-size:80%;
    margin:0px;
    color:#0f0f0f;
}
```

```
body{
    padding:50px 0px 0px 0px;
    z-index:0;
}
.navbar-dark {
     background-color: @background-dark;
}
.tab-content {
    border-left: 1px solid @lt-gray;
    border-right: 1px solid @lt-gray;
    border-bottom: 1px solid @lt-gray;
```

Note the use of the variables and the mixin with various parameters in defining the classes.

Nesting Selectors

- Next we add a carousel class to illustrate the use of nesting of classes in Less, as follows:

```
.carousel {
    background:@background-dark;


    .carousel-item {
        height: @carousel-item-height;
        img {
            position: absolute;
            top: 0;
            left: 0;
            min-height: 300px;
```

```
    }
  }
}
#carouselButton {
  right:0px;
  position: absolute;
  bottom: 0px;
  z-index: 1;
}
```

Installing and using the lessc Compiler

- Now we install the node module to support the compilation of the Less file. To do this, type the following at the command prompt:

```
npm install -g less@2.7.2
```

This will install the *less* NPM module globally so that it can be used by any project. **Note: if you are executing this on a Mac or Linux machine, you may need to add "sudo" to the beginning of this command**. This will make available the *lessc* compiler for us so that we can compile Less files.

- Next, go to the CSS folder on your machine and rename the *styles.css* file that you have there as *styles-old.css*. This is to save the CSS file that we have been using so far. We will be creating a new *styles.css* file by compiling the Less file.
- Next type the following at the command prompt to compile the Less file into a CSS file:

```
lessc styles.less styles.css
```

- You can now do a Git commit with the message "Less".

Conclusions

In this exercise you learnt to write Less code and then automatically generating the CSS file by compiling the Less code.

2.2 - Exercise (Instructions): Scss

Objectives and Outcomes

In this exercise you will learn to write Scss code and then automatically transform it into the corresponding CSS code. At the end of this exercise you will be able to:

- Write Scss code using many of the features of Scss
- Automatically convert the Scss code into CSS

Adding Scss Variables

- Open the *conFusion* project in a text editor of your choice. In the css folder, create a file named *styles.scss*. We will add the Scss code into this file.
- Add the following Scss variables into the file:

```
$lt-gray: #ddd;

$background-dark: #512DA8;

$background-light: #9575CD;

$background-pale: #D1C4E9;



// Height variables

$carousel-item-height: 300px;
```

We have just added a few color and a height variable. We will make use of these variables while defining the classes.

Scss Mixins

- Next we add a mixin into the file as follows:

```
@mixin zero-margin($pad-up-dn, $pad-left-right) {

    margin:0px auto;

    padding: $pad-up-dn $pad-left-right;

}
```

We will make use of this to define several row classes next.

- Using the variables and Mixin class that we defined earlier, add the following row classes to the file:

```
.row-header{
    @include zero-margin(0px,0px);
}
.row-content {
    @include zero-margin(50px,0px);
    border-bottom: 1px ridge;
    min-height:400px;
}
.footer{
    background-color: $background-pale;
    @include zero-margin(20px, 0px);
}
.jumbotron {
    @include zero-margin(70px,30px);
    background: $background-light ;
    color:floralwhite;
}
address{
    font-size:80%;
    margin:0px;
    color:#0f0f0f;
}
```

```
body{
    padding:50px 0px 0px 0px;
    z-index:0;
}
.navbar-dark {
     background-color: $background-dark;
}
.tab-content {
    border-left: 1px solid $lt-gray;
    border-right: 1px solid $lt-gray;
    border-bottom: 1px solid $lt-gray;
```

Note the use of the variables and the mixin with various parameters in defining the classes.

Nesting Selectors

- Next we add a carousel class to illustrate the use of nesting of classes in Scss, as follows:

```
.carousel {
    background:$background-dark;

    .carousel-item {
        height: $carousel-item-height;
        img {
            position: absolute;
            top: 0;
            left: 0;
```

```
            min-height: 300px;
        }
    }
}
#carouselButton {
    right:0px;
    position: absolute;
    bottom: 0px;
    z-index: 1;
}
```

Installing and using the node-sass module

- Now we install the node module to support the compilation of the Scss file to a CSS file. To do this, type the following at the command prompt:

```
npm install --save-dev node-sass@4.7.2
```

This will install the *node-sass* NPM module into your project and also add it as a development dependency in your package.json file.

- Next open your package.json file and add the following line into the scripts object there. This adds a script to enable the compilation of the Scss file into a CSS file:

```
"scss": "node-sass -o css/ css/"
```

- In order to transform the Scss file to a CSS file, type the following at the prompt:

```
npm run scss
```

- You can now do a Git commit with the message "Sass".

Conclusions

In this exercise you learnt to write Scss code and then automatically generating the CSS file by compiling the Scss code.

2.3 - CSS Preprocessors: Additional Resources

PDFs of Presentations

2-CSS-Preprocessors.pdf

Less and Sass Resources

- Less Getting Started
- Sass Basics
- Getting Started with Less Tutorial
- Getting Started with Sass Tutorial
- Less NPM package
- Node-sass NPM package

Assignment 4: Additional Resources

Bootstrap Documentation

- Modals
- Modal Methods

3.1 - Exercise (Instructions): NPM Scripts Part 1

Objectives and Outcomes

In this exercise, you will learn to set up NPM scripts by modifying the *package.json* file. At the end of this exercise, you will be able to:

- Watch for changes to the *styles.scss* file and automatically compile it to the css file.
- Run multiple NPM scripts in parallel using parallelshell NPM module.

Moving JS to Script file

- Create a folder named *js* and in that folder create a file named *scripts.js*.
- Open *index.html* and from this file cut out all the JQuery script that we added to it and move the code to the *scripts.js* file that we created above.
- Then, update the *index.html* file to include the *scripts.js* file by adding the following line:

```
<script src="js/scripts.js"></script>
```

- Add the same line to the scripts block in *aboutus.html* and *contactus.html*:

Watching for Changes and Parallelshell

- First, we install two NPM packages *onchange* and *parallelshell* as follows:

```
npm install --save-dev onchange@3.3.0 parallelshell@3.0.2
```

- Then, add the following two script items to *package.json* if you are doing the exercise on a MacOS computer or a Linux computer:

```
"watch:scss": "onchange 'css/*.scss' -- npm run scss",

"watch:all": "parallelshell 'npm run watch:scss' 'npm run lite'"
```

- **NOTE:** If you are doing the exercise on a Windows computer, please use the following two script items instead of the above:

```
"watch:scss": "onchange \"css/*.scss\" -- npm run scss",

"watch:all": "parallelshell \"npm run watch:scss\" \"npm run lite\""
```

- You will also update the start script as follows:

```
  "start": "npm run watch:all",
```

- Then, type the following at the prompt to start watching for changes to the SCSS file, compile it to CSS, and run the server:

```
npm start
```

- Now, whenever you make any changes to *styles.scss* file, it will automatically be compiled to the corresponding css file.
- Do a Git Commit with the message "NPM Scripts Part 1".

Conclusions

In this exercise, you learnt how to set up a watch task to watch for changes to a file and automatically run tasks upon detecting changes.

3.2 - Exercise (Instructions): NPM Scripts Part 2

Objectives and Outcomes

In this exercise you will learn to build a distribution folder containing the files that can be deployed on a web server hosting your project. This distribution folder would be built from your project files using various NPM packages and scripts. At the end of this exercise, you will be able to:

- Clean out a folder using the clean NPM module.
- Copy files from one folder to another
- Prepare a minified and concatenated css file from all the css files used in your project
- Prepare an uglified and concatenated JS file containing all the JS code used in your project

Cleaning up a Distribution Folder

- Install the *rimraf* npm module by typing the following at the prompt:

```
npm install --save-dev rimraf@2.6.2
```

- Then, set up the following script:

```
  "clean": "rimraf dist",
```

Copying Fonts

- Your project uses font-awesome fonts. These need to be copied to the distribution folder. To help us do this, install the *copyfiles* NPM module globally as follows:

```
npm -g install copyfiles@2.0.0
```

Remember to use *sudo* on mac and Linux.

- Then set up the following script:

```
  "copyfonts": "copyfiles -f node_modules/font-awesome/fonts/* dist/fonts",
```

Compressing and Minifying Images

- We use the *imagemin-cli* NPM module to help us to compress our images to reduce the size of the images being used in our project. Install the *imagemin-cli* module as follows:

```
npm -g install imagemin-cli@3.0.0
```

Remember to use *sudo* on mac and Linux. **NOTE:** Some students have encountered issues with imagemin-cli not installing its plugins due to issues with global permissions on Mac. In that case try

```
sudo npm install -g imagemin-cli@3.0.0 --unsafe-perm=true --allow-root
```

- Then set up the following script:

```
  "imagemin": "imagemin img/* --out-dir='dist/img'",
```

Preparing the Distribution Folder

- Open *.gitignore* and update it as follows. We do not want the dist folder to be checked into the git repository.

```
node_modules

dist
```

- Then, install the *usemin-cli, cssmin, uglifyjs* and *htmlmin* NPM packages as follows:

```
npm install --save-dev usemin-cli@0.5.1 cssmin@0.4.3 uglifyjs@2.4.11 htmlmin@0.0.7
```

- Add the following two scripts to the package.json file:

```
  "usemin": "usemin contactus.html -d dist --htmlmin -o dist/contactus.html && usemin aboutus.html -d dist --htmlmin -o dist/aboutus.html && usemin index.html -d dist --htmlmin -o dist/index.html",

  "build": "npm run clean && npm run imagemin && npm run copyfonts && npm run usemin"
```

- Open *index.html* and surround the css links inclusion code as follows:

```
  <!-- build:css css/main.css -->

  <link rel="stylesheet" href="node_modules/bootstrap/dist/css/bootstrap.min.css">

  <link rel="stylesheet" href="node_modules/font-awesome/css/font-awesome.min.css">

  <link rel="stylesheet" href="node_modules/bootstrap-social/bootstrap-social.css">

  <link href="css/styles.css" rel="stylesheet">

  <!-- endbuild -->
```

- Do the same change in *aboutus.html* and *contactus.html*

- Similarly, open *index.html* and surround the js script inclusion code as follows:

```
<!-- build:js js/main.js -->

<script src="node_modules/jquery/dist/jquery.slim.min.js"></script>

<script src="node_modules/popper.js/dist/umd/popper.min.js"></script>

<script src="node_modules/bootstrap/dist/js/bootstrap.min.js"></script>

<script src="js/scripts.js"></script>

<!-- endbuild -->
```

- Do the same change in *aboutus.html* and *contactus.html*
- To build the distribution folder, you can type the following at the prompt:

```
npm run build
```

- This will build the *dist* folder containing the files that are a self-contained version of your project. You can now copy the contents of this folder to a web server that hosts your website.
- After verifying that the dist folder is built correctly, you can now do a git commit with the message "NPM Scripts Part 2"

Conclusions

In this exercise, you learnt the various steps to build the project for deployment using NPM scripts.

3.3 - Building and Deployment: NPM Scripts: Additional Resources

PDFs of Presentations

3-Building-Deployment.pdf

4-NPM-Scripts.pdf

Additional Resources

- Why npm Scripts?
- How to Use npm as a Build Tool
- The Command Line for Web Design

NPM Modules

- onchange
- parallelshell
- rimraf
- copyfiles
- imagemin-cli
- usemin-cli
- cssmin
- uglifyjs
- htmlmin

4.1 - Exercise (Instructions): Grunt Part 1

Objectives and Outcomes

In this exercise, you will learn to use Grunt, the task runner. You will install Grunt CLI and install Grunt packages using NPM. Thereafter you will configure a Grunt file with a set of tasks to build and serve your web project. At the end of this exercise, you will be able to:

- Install Grunt CLI and Grunt packages in your project
- Configure a Grunt file with a set of tasks to build a web project from a source, and serve the built project using a server.

Installing Grunt

- At the command prompt, type the following to install Grunt command-line interface (CLI):

```
npm install -g grunt-cli@1.2.0
```

This will install the Grunt CLI globally so that you can use them in all projects.

- Next install Grunt to use within your project. To do this, go to the *conFusion* folder and type the following at the prompt:

```
npm install grunt@1.0.2 --save-dev
```

This will install local per-project Grunt to use within your project.

Creating a Grunt File

- Next you need to create a Grunt file containing the configuration for all the tasks to be run when you use Grunt. To do this, create a file named *Gruntfile.js* in the *conFusion* folder.
- Next, add the following code to Gruntfile.js to set up the file to configure Grunt tasks:

```
'use strict';

module.exports = function (grunt) {

  // Define the configuration for all the tasks

  grunt.initConfig({

  });

};
```

This sets up the Grunt module ready for including the grunt tasks inside the function above.

Compiling SCSS to CSS

- Next, we are going to set up our first Grunt task. The SASS task converts the SCSS code to CSS. To do this, you need to include some Grunt modules that help us with the tasks. Install the following modules by typing the following at the prompt:

```
npm install grunt-sass@2.1.0 --save-dev

npm install time-grunt@1.4.0 --save-dev

npm install jit-grunt@0.10.0 --save-dev
```

The first one installs the Grunt module for SCSS to CSS conversion. The time-grunt module generates time statistics about how much time each task consumes, and jit-grunt enables us to include the necessary downloaded Grunt modules when needed for the tasks.

- Now, configure the SASS task in the Gruntfile as follows, by including the code inside the function in *Gruntfile.js*:

```
'use strict';

module.exports = function (grunt) {

  // Time how long tasks take. Can help when optimizing build times

  require('time-grunt')(grunt);



  // Automatically load required Grunt tasks

  require('jit-grunt')(grunt);



  // Define the configuration for all the tasks

  grunt.initConfig({

    sass: {

      dist: {

        files: {

          'css/styles.css': 'css/styles.scss'
```

```
            }

        }

    }

});


grunt.registerTask('css', ['sass']);

};
```

- Now you can run the grunt SASS task by typing the following at the prompt:

```
grunt css
```

Watch and Serve Tasks

- The final step is to use the Grunt modules watch and browser-sync to spin up a web server and keep a watch on the files and automatically reload the browser when any of the watched files are updated. To do this, install the following grunt modules:

```
npm install grunt-contrib-watch@1.0.0 --save-dev

npm install grunt-browser-sync@2.2.0 --save-dev
```

- After this, we will configure the browser-sync and watch tasks by adding the following code to the Grunt file:,

```
    watch: {

        files: 'css/*.scss',

        tasks: ['sass']

    },

    browserSync: {

        dev: {

            bsFiles: {

                src : [

                    'css/*.css',
```

```
                '*.html',
                'js/*.js'
            ]
        },
        options: {
            watchTask: true,
            server: {
                baseDir: "./"
            }
        }
    }
}
```

- Then add the following task to the Grunt file:

```
grunt.registerTask('default', ['browserSync', 'watch']);
```

- Now if you type the following at the command prompt, it will start the server, and open the web page in your default browser. It will also keep a watch on the files in the css folder, and if you update any of them, it will compile the scss file into css file and load the updated page into the browser (livereload)

```
grunt
```

- Do a Git commit with the message "Grunt Part 1".

Conclusions

In this exercise you have learnt how to configure a Grunt file to perform several tasks. You were able to start a server with livereload to serve the web page.

4.2 - Exercise (Instructions): Grunt Part 2

Objectives and Outcomes

In this exercise, you will continue to learn to use Grunt, the task runner. You will configure the Grunt file with a set of additional tasks to build your web project. At the end of this exercise, you will be able to:

- Configure a Grunt file with a set of tasks to build your web project from a source.

Copying the Files and Cleaning Up the Dist Folder

- Next you will install the Grunt modules to copy over files to a distribution folder named dist, and clean up the dist folder when needed. To do this, install the following Grunt modules:

```
npm install grunt-contrib-copy@1.0.0 --save-dev

npm install grunt-contrib-clean@1.1.0 --save-dev
```

- You will now add the code to perform the copying of files to the dist folder, and cleaning up the dist folder. To do this, add the following code to *Gruntfile.js*. This should be added right after the configuration of the SASS task.:

```
,

    copy: {

      html: {

        files: [

        {

          //for html

          expand: true,

          dot: true,

          cwd: './',

          src: ['*.html'],

          dest: 'dist'

        }]
```

```
    },
    fonts: {
        files: [
        {
            //for font-awesome
            expand: true,
            dot: true,
            cwd: 'node_modules/font-awesome',
            src: ['fonts/*.*'],
            dest: 'dist'
        }]
    }
},


clean: {
    build: {
        src: [ 'dist/']
    }
}
```

- Remember to add the comma after the end of the SASS task.

Compressing and Minifying Images

- Next we install the grunt-contrib-imagemin module and use it to process the images. To install this module type at the prompt:

```
npm install grunt-contrib-imagemin@2.0.1 --save-dev
```

- Then, configure the imagemin task as shown below in the Gruntfile:

```
,

    imagemin: {

      dynamic: {

        files: [{

          expand: true,                  // Enable dynamic expansion

          cwd: './',                   // Src matches are relative to this path

          src: ['img/*.{png,jpg,gif}'],   // Actual patterns to match

          dest: 'dist/'                  // Destination path prefix

        }]

      }

    }
```

Preparing the Distribution Folder and Files

- We are now going to use the Grunt *usemin* module together with *concat*, *cssmin*, *uglify* and *filerev* to prepare the distribution folder. To do this, install the following Grunt modules:

```
npm install grunt-contrib-concat@1.0.1 --save-dev

npm install grunt-contrib-cssmin@2.2.1 --save-dev

npm install grunt-contrib-htmlmin@2.4.0 --save-dev

npm install grunt-contrib-uglify@3.3.0 --save-dev

npm install grunt-filerev@2.3.1 --save-dev

npm install grunt-usemin@3.1.1 --save-dev
```

- Next, update the task configuration within the Gruntfile.js with the following additional code to introduce the new tasks:

```
,

    useminPrepare: {

        foo: {

            dest: 'dist',

            src: ['contactus.html','aboutus.html','index.html']

        },

        options: {

            flow: {

                steps: {

                    css: ['cssmin'],

                    js:['uglify']

                },

                post: {

                    css: [{

                        name: 'cssmin',

                        createConfig: function (context, block) {

                        var generated = context.options.generated;

                            generated.options = {

                                keepSpecialComments: 0, rebase: false

                            };

                        }

                    }]

                }
```

```
            }
        }
    },
    // Concat
    concat: {
        options: {
            separator: ';'
        },
        // dist configuration is provided by useminPrepare
        dist: {}
    },

    // Uglify
    uglify: {
```

- Next, update the jit-grunt configuration as follows, to inform it that useminPrepare task depends on the usemin package:

```
require('jit-grunt')(grunt, {
  useminPrepare: 'grunt-usemin'
});
```

- Next, update the Grunt build task as follows:

```
grunt.registerTask('build', [

    'clean',

    'copy',

    'imagemin',

    'useminPrepare',

    'concat',

    'cssmin',

    'uglify',

    'filerev',

    'usemin',

    'htmlmin'

]);
```

- Now if you run Grunt, it will create a dist folder with the files structured correctly to be distributed to a server to host your website. To do this, type the following at the prompt:

```
grunt build
```

Conclusions

In this exercise you have learnt how to configure a Grunt file to perform several tasks. You were able to build a distribution folder for your web project.

4.3 - Exercise (Instructions): Gulp Part 1

Objectives and Outcomes

In this exercise, you will learn to use Gulp, the task runner. You will install Gulp CLI and install Gulp plugins using NPM. Thereafter you will configure a Gulp file with a set of tasks to build and serve your web project. At the end of this exercise, you will be able to:

- Install Gulp CLI and Gulp plugins in your project
- Configure a Gulp file with a set of tasks to build a web project from a source, and serve the built project using a server.

Installing Gulp

- At the command prompt, type the following to install Gulp command-line interface (CLI) globally:

```
npm install -g gulp-cli@2.0.1
```

This will install the Gulp globally so that you can use it in all projects.

- Next install Gulp to use within your project. To do this, go to the *conFusion* folder and type the following at the prompt:

```
npm install gulp@3.9.1 --save-dev
```

This will install local per-project Gulp to use within your project.

Install Gulp Plugins for SASS and Browser-Sync

- Install all the Gulp plugins that you will need for this exercise. To do this, type the following at the command prompt:

```
npm install gulp-sass@3.1.0  browser-sync@2.23.6 --save-dev
```

Creating a Gulp File

- Next you need to create a Gulp file containing the tasks to be run when you use Gulp. To do this, create a file named *gulpfile.js* in the *conFusion* folder.

Loading Gulp Plugins

- Load in all the Gulp plugins by including the following code in the Gulp file:

```
'use strict';

var gulp = require('gulp'),

    sass = require('gulp-sass'),

    browserSync = require('browser-sync');
```

Adding Gulp Tasks for SASS and Browser-Sync

- Next, we will add the code for the SASS task, the Browser-Sync task and the default task as follows:

```
gulp.task('sass', function () {

  return gulp.src('./css/*.scss')

    .pipe(sass().on('error', sass.logError))

    .pipe(gulp.dest('./css'));

});



gulp.task('sass:watch', function () {

  gulp.watch('./css/*.scss', ['sass']);

});



gulp.task('browser-sync', function () {

   var files = [

      './*.html',

      './css/*.css',

      './img/*.{png,jpg,gif}',
```

```
    './js/*.js'
  ];


  browserSync.init(files, {
    server: {
      baseDir: "./"
    }
  });
});


// Default task
gulp.task('default', ['browser-sync'], function() {
   gulp.start('sass:watch');
});
```

- Save the Gulp file

Running the Gulp Tasks

- At the command prompt, if you type *gulp* it will run the default task:

```
gulp
```

- Do a Git commit with the message "Gulp Part 1".

Conclusions

In this exercise, you learnt to use Gulp, install Gulp plugins, configure the gulpfile.js and then use Gulp to automate the web development tasks.

4.4 - Exercise (Instructions): Gulp Part 2

Objectives and Outcomes

In this exercise, you will continue to learn to use Gulp. Thereafter you will configure a Gulp file with a set of tasks to build and serve your web project. At the end of this exercise, you will be able to:

- Configure the Gulp file with a set of tasks to build the distribution folder for the web project.

Copying the Files and Cleaning up the Dist Folder

- We will now create the tasks for copying the font files and cleaning up the distribution folder. To do this we will first install the *del* Node module and require it in the Gulp file as follows:

```
npm install del@3.0.0 --save-dev
```

```
var ...
    del = require('del'),
    ...
```

- Next, we will add the code for the Clean task and the copyfonts task as follows:

```
// Clean
gulp.task('clean', function() {
    return del(['dist']);
});
gulp.task('copyfonts', function() {
   gulp.src('./node_modules/font-awesome/fonts/**/*.{ttf,woff,eof,svg}*')
   .pipe(gulp.dest('./dist/fonts'));
});
```

Compressing and Minifying Images

- We will now install the *gulp-imagemin* plugin and configure the *imagemin* task. To do this we install the plugin and require it as follows:

```
npm install gulp-imagemin@4.1.0 --save-dev



var ...

    imagemin = require('gulp-imagemin'),

    ...
```

- Next, we create the *imagemin* task as follows:

```
// Images

gulp.task('imagemin', function() {

  return gulp.src('img/*.{png,jpg,gif}')

    .pipe(imagemin({ optimizationLevel: 3, progressive: true, interlaced: true }))

    .pipe(gulp.dest('dist/img'));

});
```

Preparing the Distribution Folder and Files

- We now install the gulp-usemin and other related Gulp plugins and require them as follows:

```
npm install gulp-uglify@3.0.0 gulp-usemin@0.3.29 gulp-rev@8.1.1 gulp-clean-css@3.9.3 gulp-flatmap@
1.0.2 gulp-htmlmin@4.0.0 --save-dev



var ...

    uglify = require('gulp-uglify'),

    usemin = require('gulp-usemin'),

    rev = require('gulp-rev'),

    cleanCss = require('gulp-clean-css'),
```

```
    flatmap = require('gulp-flatmap'),
    htmlmin = require('gulp-htmlmin');
```

- We configure the usemin and the build task as follows:

```
gulp.task('usemin', function() {
  return gulp.src('./*.html')
  .pipe(flatmap(function(stream, file){
      return stream
        .pipe(usemin({
            css: [ rev() ],
            html: [ function() { return htmlmin({ collapseWhitespace: true })} ],
            js: [ uglify(), rev() ],
            inlinejs: [ uglify() ],
            inlinecss: [ cleanCss(), 'concat' ]
        }))
    }))
    .pipe(gulp.dest('dist/'));
});


gulp.task('build',['clean'], function() {
    gulp.start('copyfonts','imagemin','usemin');
});
```

- Save the Gulp file

Running the Gulp Tasks

- At the command prompt, if you type *gulp build* it will run the build task:

```
gulp build
```

- Do a Git commit with the message "Gulp Part 2"

Conclusions

In this exercise, you learnt to use Gulp, install Gulp plugins, configure the gulpfile.js and then use Gulp to automate the web development tasks.

4.5 - Building and Deployment: Task Runners: Additional Resources

PDFs of Presentations

5-Task-Runners.pdf

Grunt Resources

- Grunt
- Writing an Awesome Build Script with Grunt
- Clean Grunt
- File Globbing
- The Command Line for Web Design: Automation With Grunt

Grunt Plugins

- grunt-contrib-jshint
- jshint-stylish
- grunt-contrib-copy
- grunt-contrib-clean
- grunt-usemin
- grunt-contrib-concat
- grunt-contrib-cssmin
- grunt-contrib-htmlmin
- grunt-contrib-uglify
- grunt-filerev

Gulp Resources

- Gulp
- An Introduction to Gulp.js
- Getting started with gulp
- Building with Gulp
- The Command Line for Web Design: Automation with Gulp

Gulp Plugins

- gulp
- gulp-sass
- browser-sync
- del
- gulp-imagemin
- gulp-uglify
- gulp-usemin
- gulp-rev
- gulp-clean-css
- gulp-flatmap
- gulp-htmlmin

Tasks

- Minification
- UglifyJS
- JSHint

General Resources

- Node, Grunt, Bower and Yeoman - A Modern web dev's Toolkit
- The Advantages of Using Task Runners
- Gulp vs Grunt. Why one? Why the Other?
- Why we should stop using Grunt & Gulp
- Why I Left Gulp and Grunt for npm Scripts

Front-End Web UI Frameworks: Bootstrap 4: Conclusions

PDFs of Presentations

7-Conclusion.pdf