Exercise (Instructions): Web Tools: Grunt

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

Note: You should already have Node and NPM installed on your computer before you proceed further. Also, those using OSX or Linux should use **sudo** while installing **global** packages in node (when you use the **-g** flag).

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

```
    npm install -g grunt-cli
```

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

- Next, create the *package.json* file in the *conFusion* folder with the following content:

```
{
  "name": "conFusion",
  "private": true,
  "devDependencies": {

  },
  "engines": {
    "node": ">=0.10.0"
  }
}
```

- 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 --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.

Configuring the Project

- Next, we need to configure the menu.html file for this exercise. Open the menu.html file and update the CSS import lines in the head of the page as follows:

```
        <!-- Bootstrap -->
<!-- build:css styles/main.css -->
    <link href="../bower_components/bootstrap/dist/css/bootstrap.min.css" rel="stylesheet
">
    <link href="../bower_components/bootstrap/dist/css/bootstrap-theme.min.css" rel="styl
esheet">
    <link href="../bower_components/font-awesome/css/font-awesome.min.css" rel="styleshee
t">
    <link href="styles/bootstrap-social.css" rel="stylesheet">
    <link href="styles/mystyles.css" rel="stylesheet">
<!-- endbuild -->
```

We are surrounding the links with some specially formatted comment lines, the purpose of which will become clear in a later step.

- Similarly, replace all the script at the bottom of the page with the following code. We will be moving all the Angular script to a separate *app.js* file in the next step.

```
<!-- build:js scripts/main.js -->
    <script src="../bower_components/angular/angular.min.js"></script>
    <script src="scripts/app.js"></script>
<!-- endbuild -->
```

- Next, in the app/scripts folder, create a file named *app.js* and include the following code in this file:

```
'use strict';
angular.module('confusionApp', [])


        .controller('menuController', function() {
                        this.tab = 1;
            this.filtText = '';
            var dishes=[
                         {
                          name:'Uthapizza',
                          image: 'images/uthapizza.png',
                          category: 'mains',
                          label:'Hot',
                          price:'4.99',
                          description:'A unique combination of Indian Uthappam (pancake)
and Italian pizza, topped with Cerignola olives, ripe vine cherry tomatoes, Vidalia onion
, Guntur chillies and Buffalo Paneer.',
                           comment: ''
                        },
                        {
                           name:'Zucchipakoda',
                           image: 'images/zucchipakoda.png',
                           category: 'appetizer',
                           label:'',
                           price:'1.99',
                          description:'Deep fried Zucchini coated with mildly spiced Chic
kpea flour batter accompanied with a sweet-tangy tamarind sauce',
                           comment: ''
```

```
                    },
                    {
                        name:'Vadonut',
                        image: 'images/vadonut.png',
                        category: 'appetizer',
                        label:'New',
                        price:'1.99',
                       description:'A quintessential ConFusion experience, is it a vad
a or is it a donut?',
                        comment: ''
                    },
                    {
                        name:'ElaiCheese Cake',
                        image: 'images/elaicheesecake.png',
                       category: 'dessert',
                        label:'',
                       price:'2.99',
                       description:'A delectable, semi-sweet New York Style Cheese Cak
e, with Graham cracker crust and spiced with Indian cardamoms',
                        comment: ''
                    }
                    ];
            this.dishes = dishes;

            this.select = function(setTab) {
               this.tab = setTab;
               if (setTab === 2) {
                   this.filtText = "appetizer";
               }
               else if (setTab === 3) {
                   this.filtText = "mains";
               }
               else if (setTab === 4) {
                   this.filtText = "dessert";
               }
               else {
                   this.filtText = "";
               }
```

```
        };
        this.isSelected = function (checkTab) {
            return (this.tab === checkTab);
        };
    });
```

With this change we have moved all the JavaScript code into a separate file. This is the normal way that you configure Angular projects.

The JSHint Task

- Next, we are going to set up our first Grunt task. The JSHint task validates our JavaScript code and points out errors and gives warnings about minor violations. 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-contrib-jshint --save-dev
npm install jshint-stylish --save-dev
npm install time-grunt --save-dev
npm install jit-grunt --save-dev
```

The first one installs the Grunt module for JSHint, and the second one adds further to print out the messages from JSHint in a better format. 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 JSHint task in the Gruntfile as follows, by including the code inside the function in*Gruntfile.js*:

```
// 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({
        pkg: grunt.file.readJSON('package.json'),

  // Make sure code styles are up to par and there are no obvious mistakes
  jshint: {
    options: {
      jshintrc: '.jshintrc',
```

```
        reporter: require('jshint-stylish')
      },
      all: {
        src: [
          'Gruntfile.js',
          'app/scripts/{,*/}*.js'
        ]
      }
    }
  });

  grunt.registerTask('build', [
    'jshint'
  ]);

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

In the above, we have set up time-grunt and jit-grunt to time the tasks, and load Grunt modules when needed. The JSHint task is set to examine all the JavaScript files in the app/scripts folder, and the Gruntfile.js and generate any reports of JS errors or warnings. We have set up a Grunt task named build, that runs the jshint task and the build task is also registered as the default task.

- Next, create a file named *.jshintrc* (Don't forget the . in front of jshintrc) in the conFusion folder, and include the following in the file:

```
{
  "bitwise": true,
  "browser": true,
  "curly": true,
  "eqeqeq": true,
  "esnext": true,
  "latedef": true,
  "noarg": true,
  "node": true,
  "strict": true,
  "undef": true,
  "unused": true,
  "globals": {
    "angular": false
```

```
  }
}
```

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

```
    grunt
```

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 --save-dev
    npm install grunt-contrib-clean --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 JSHint task.:

```
  },
  copy: {
    dist: {
      cwd: 'app',
      src: [ '**','!styles/**/*.css','!scripts/**/*.js' ],
      dest: 'dist',
      expand: true
    },
    fonts: {
        files:[
            {
                //for bootstrap fonts
                  expand: true,
                  dot: true,
                  cwd: 'bower_components/bootstrap/dist',
                  src: ['fonts/*.*'],
                  dest: 'dist'
              }, {
                  //for font-awesome
                  expand: true,
```

```
                dot: true,
                cwd: 'bower_components/font-awesome',
                src: ['fonts/*.*'],
                dest: 'dist'
            }
        ]
    }
  },
  clean: {
      build:{
          src: [ 'dist/']
      }
  }
```

- Then, update the Grunt build task in the file as follows:

```
grunt.registerTask('build', [
  'clean',
  'jshint',
  'copy'
]);
```

- You can run Grunt and see what it generates.

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 --save-dev
npm install grunt-contrib-cssmin --save-dev
npm install grunt-contrib-uglify --save-dev
npm install grunt-filerev --save-dev
npm install grunt-usemin --save-dev
```

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

```
    },
    useminPrepare: {
        html: 'app/menu.html',
        options: {
            dest: 'dist'
        }
    },
      // Concat
    concat: {
        options: {
            separator: ';'
        },
        // dist configuration is provided by useminPrepare
        dist: {}
    },
      // Uglify
    uglify: {
        // dist configuration is provided by useminPrepare
        dist: {}
    },
    cssmin: {
        dist: {}
    },
      // Filerev
    filerev: {
        options: {
            encoding: 'utf8',
            algorithm: 'md5',
            length: 20
        },
        release: {
            // filerev:release hashes(md5) all assets (images, js and css )
            // in dist directory
            files: [{
                src: [
                    'dist/scripts/*.js',
                    'dist/styles/*.css',
```

```
            ]
        }]
    }
  },
    // Usemin
    // Replaces all assets with their revved version in html and css files.
    // options.assetDirs contains the directories for finding the assets
    // according to their relative paths
  usemin: {
      html: ['dist/*.html'],
      css: ['dist/styles/*.css'],
      options: {
          assetsDirs: ['dist', 'dist/styles']
      }
  },
```

- 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',
  'jshint',
  'useminPrepare',
  'concat',
  'cssmin',
  'uglify',
  'copy',
  'filerev',
  'usemin'
]);
```

- 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.

Watch, Connect and Serve Tasks

- The final step is to use the Grunt modules watch, connect and watch, 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 --save-dev
npm install grunt-contrib-connect --save-dev
```

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

```
watch: {
    copy: {
        files: [ 'app/**', '!app/**/*.css', '!app/**/*.js'],
        tasks: [ 'build' ]
    },
    scripts: {
        files: ['app/scripts/app.js'],
        tasks:[ 'build']
    },
    styles: {
        files: ['app/styles/mystyles.css'],
        tasks:['build']
    },
    livereload: {
        options: {
            livereload: '<%= connect.options.livereload %>'
        },
        files: [
            'app/{,*/}*.html',
            '.tmp/styles/{,*/}*.css',
            'app/images/{,*/}*.{png,jpg,jpeg,gif,webp,svg}'
        ]
  }
},
```

```
    connect: {
      options: {
        port: 9000,
        // Change this to '0.0.0.0' to access the server from outside.
        hostname: 'localhost',
        livereload: 35729
      },
      dist: {
        options: {
          open: true,
          base:{
               path: 'dist',
            options: {
                index: 'menu.html',
                maxAge: 300000
            }
          }
        }
      }
    },
```

- Then add the following task to the Grunt file:

```
 grunt.registerTask('serve',['build','connect:dist','watch']);
```

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

```
    grunt serve
```

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 and start a server with livereload to serve the web page.