Exercise (Instructions): Angular Scope

Objectives and Outcomes

In this exercise, you will explore Angular scope and the use of scope as a glue between the view and controller. You will also learn about the *ngShow* directive. At the end of this exercise, you will be able to:

- Understand the use of Angular Scope.
- Use scope to connect the view and the controller.
- Make use of the ngShow directive.

Modifying the Gulp File

- Since Angular involves writing a lot of JavaScript, once we introduce the $scope, we need to make sure that when the uglify task runs, it does not end up mangling the $scope. Otherwise the JavaScript code will not work. Fortunately, we have an gulp plugin named gulp-ng-annotate, which ensures the mangling does not cause any problems. We now need to add this plugin and update the gulpfile.js to include this plugin.
- First install the gulp-ng-annotate plugin:

```
npm install gulp-ng-annotate --save-dev
```

- Then require this in gulpfile.js.

```
var ngannotate = require('gulp-ng-annotate');
```

- Next, add the ngannotate() to the usemin task for the JavaScript part, by updating the usemin task as follows:

```
gulp.task('usemin',['jshint'], function () {
  return gulp.src('./app/menu.html')
      .pipe(usemin({
        css:[minifycss(),rev()],
        js: [ngannotate(),uglify(),rev()]
      }))
      .pipe(gulp.dest('dist/'));
});
```

Using Angular $Scope

- Open the *app.js* file. Update the Angular controller's name to MenuController, changing the small letter "m" to capital letter "M", to conform to the accepted Angular convention of naming the controllers starting with a Capital letter.
- Next, update the controller to use the scope as follows:

```
        .controller('MenuController', ['$scope', function($scope) {
           . . .
        }]);
```

- Next, you need to update all references to "*this.*" with "*$scope.*" in the JavaScript code in the controller.
- **Remove** the following statement from the controller code:

```
    this.dishes = dishes;
```

- In its place, substitute the following statement:

```
    $scope.dishes = dishes;
```

- Save the *app.js* file, and then open *menu.html* file. In the HTML code, we no longer need to use the *menuCtrl* alias for the *MenuController*. The JavaScript variables and functions in the *MenuController* code can be accessed within HTML by directly using their names without the *menuCtrl.* prefix. So, remove the *menuCtrl.* prefix from all the HTML code. Also remove the *menuCtrl* alias from the ng-controller directive. Also, update the *menuController* to *MenuController* in the directive.

Using the *ngShow* Directive

- In the *menu.html* page, right before the <ul> for the tabs, introduce a button using the following code:

```
                <button ng-click="toggleDetails()" class="btn btn-xs btn
                  -primary pull-right"                    type="button">
                   {{showDetails ? 'Hide Details':'Show Details'}}
                </button>
```

- Update the <p> containing the dish description as follows:

```
                        <p ng-show="showDetails">{{dish.description}}</p>
```

- Save the *menu.html* page, and then switch to *app.js* file to introduce the JavaScript code. Add the following code to the *MenuController*:

```
            $scope.showDetails = false;

            . . .

            $scope.toggleDetails = function() {
                $scope.showDetails = !$scope.showDetails;
            };
```

- Save *app.js* and then check the behavior of the web page.

Conclusions

In this exercise, you learnt more about the use of Angular $Scope. You also learnt about the *ngShow* directive.

Mark as completed