PGCert IT: Programming with Web Technologies

Lab 13 – Handlebars

Before you Start

Note that there is no BitBucket repository for today's lab. You'll be creating everything from scratch.

Exercise One: Example

For the first exercise, we'll be looking at the provided example code, making sure it runs, and answering some questions about it.

To start, download the provided examples on Canvas. You'll notice that there's a `package.json` file, but no `node_modules` folder. It is best practice not to include this folder as it takes up a fair bit of space, and can easily be recreated when necessary as long as the `package.json` file is available, using node package manager.

To do this, open a terminal window at the location of the `package.json` file, and type the following command:

```
npm install
```

Once this is done, you'll notice the folder has been recreated.

Also in the examples, you'll notice a file called `.gitignore`. Feel free to examine its contents in a text editor. This is a special file recognized by git. Any files specified in `.gitignore` will be ignored by the version control system. You'll see that it is currently configured to ignore anything in the `node_modules` folder. You should generally keep things configured this way to avoid committing unnecessary files to BitBucket. Feel free to copy this file and use it in your own projects (you should put it in the same place as your `.git` folder).

Once you've reinstalled the required packages as above, open the project in VS Code. Then, you should be able to run any of the example by opening that file in the editor window and hitting F5.

Examples one through three show off the concepts discussed in the lecture, while example four illustrates a more complex example. Study example four, then answer the following questions related to it:

1. What happens when the user navigates to http://localhost:3000/notexist/? Why? In your answer, refer to code contained within `example04.js`.

   It will render /public/error-page.html as we don't have any code to handle /notexist

2. In `views/example04/example04-about.handlebars`, what is happening on Line 12?

It is grabbing the numbers property of the data json being passed through by the "res.render('example04/example04-about', data);"

3. Explain, in as much detail as you can, what happens when the user navigates to http://localhost:3000/contact/.

The browser will render the "example04/example04-contact.handlebars" view based on the "/views/layouts/example04/example04-main.handlebars" layout. The view will then use the "data" JSON to populate elements in it's HTML code. Referring to each one using the "data" JSON's property names i.e. {{email}}. It also uses an "each" helper to add each phone number in the phoneNumbers array as a list item within the <ul> element in the table.

4. How does the code display the navbar on every page, despite the corresponding HTML only appearing in one file?

The navbar is defined in the layout file - example04-main.handlebars. This is set to the default layout by line 17 & 18 in example04.js.

```
app.engine('handlebars', handlebars({ defaultLayout: 'example04/example04-main' }));
app.set('view engine', 'handlebars');
```

5. Describe in detail how the code is ensuring that the appropriate link on the navbar appears "selected", depending on whether the user is viewing the homepage, about, or contacts section? Refer to code contained within example04-`main.handlebars` and `example04.js` in your answer.

The data JSON's used for each app.get and app.render uses a boolean to let the layout know when the currently rendered page has been loaded. Otherwise, this boolean will remain false when the page hasn't been requested.

6. How does the code display multiple articles, when the HTML to display an article only appears once within example04-`home.handlebars`?

It uses the {{#each articles}} helper to loop through each article and creates a card for each instance.

Exercise Two: Templating by Hand

Assume that the following Handlebars files have been defined:

In `views/ex02.handlebars`:

```
<p>My name is {{name}}, and I am {{{description}}}!</p>
```

In `views/layouts/main.handlebars`:

```
<!DOCTYPE html>
<html>
<head>
    <title>Handlebars – {{title}}</title>
</head>
<body>
    {{{body}}}
</body>
</html>
```

Also, assume the following code is contained within `app.js` (you may assume all necessary Express & Handlebars setup code is also in place):

```
app.get('/', function (req, res) {
    var data = {
        title: "Exercise Two",
        name: "Bob",
        description: "<em>Awesome</em>"
    };
    res.render('ex02', data);
});
```

1. What HTML would be sent to the browser when the user navigated to http://localhost:3000/?

   My name is Bob, and I am Awesome!

   ("Awesome will be italicized")

2. What would change if ex02.handlebars contained `{{description}}` instead of `{{{description}}}`?

   My name is Bob, and I am <em>Awesome</em>!

Exercise Three: From Scratch

In this exercise, you'll be creating an Express / Handlebars app from scratch.

1. To start with, create a new folder for the exercise, then open a terminal window in that folder and configure / install the necessary software using npm. Specifically, first create `package.json` using:

   ```
   npm init
   ```

   Note: After the package.json file has been created, you may need to quit the command line application using Ctrl + C.

2. Next, install Express and Handlebars using:

   ```
   npm install --save express
   npm install --save express-handlebars
   ```

3. Create a `.gitignore` file for ignoring the `node_modules` folder (or just copy the existing file in the example project).

4. Once that's done, create an `app.js` file for your code, and a `views` folder where your Handlebars files will go. In the `views` folder, also create a `layouts` folder for your Handlebars layouts. Finally, create a `public` folder where you can place static content to be served by your app.

5. Add the following code to `app.js`, which will serve as a useful starting point:

   ```
   var express = require('express');
   var fs = require('fs');

   var app = express();

   app.set('port', process.env.PORT || 3000);

   var handlebars = require('express-handlebars');
   app.engine('handlebars', handlebars({ defaultLayout: 'main' }));
   app.set('view engine', 'handlebars');

   app.use(express.static(__dirname + "/public"));

   app.listen(app.get('port'), function () {
       console.log('Express started on http://localhost:' +
         app.get('port'));
   });
   ```

   This code sets up Express, registers Handlebars as the *view engine*, allows files in the `public` folder to be served to clients, and starts the app running on port 3000. With this code, if you place any files in the public folder and try navigating to them, they should be served to the browser correctly. Attempting to navigate to any other route will result in the default Express 404 page being displayed (e.g. `Cannot GET /non/existant/path` or similar), as none have been defined.

6. Create a Handlebars *layout*, called `main.handlebars`, in the `views/layouts` folder. Layouts contain HTML which is commonly shared amongst pages, such as `<head>` information, menu bars, etc. For now, add the following code to the file:

```
<!DOCTYPE html>
<html>
<head>
    <title>My Personal Website</title>
</head>
<body>
    {{{body}}}
</body>
</html>
```

   The `{{{body}}}` Handlebars expression will be replaced with the contents of a particular *view*.

7. Create a Handlebars *view*, called `about.handlebars`, in the `views` folder. Populate this view with information about yourself (in standard HTML). Remember that you only need to supply the contents of the HTML `<body>` in this file, as the remainder of the HTML will be generated by `main.handlebars` from step 6.

8. Modify app.js so that when the user navigates to "/" with a GET request, your page is displayed. Refer to the provided example code for a reminder of how to do this if required.

9. Create two more views, called `404.handlebars` and `500.handlebars`. Have these files display appropriate messages regarding errors which have occurred (404 refers to clients trying to access non-existent files, while 500 errors refer to server problems).

10. Add the following code to the end of your `app.js` file (just before "`app.listen`…"):

```
app.use(function(req, res, next) {
    res.status(404);
    res.render('404');
});
app.use(function(err, req, res, next) {
    console.error(err.stack);
    res.status(500);
    res.render('500');
});
```

   The first function will be called whenever the client tries to navigate to any route not previously covered in the file (hence why this one should be last), and will display the 404 page. The second function will be called whenever there's a server error. It will print the error to the console for debugging purposes, and display your 500 page.

Exercise Four: Listing Hobbies

Modify your solution to exercise three, such that the page also displays a list of hobbies you like, and displays a random one of those hobbies which you like doing on the weekend. Specifically, perform the following tasks:

1. In app.js, create an array and populate it with your hobbies. For example:

   ```
   var hobbies = ["Sleeping", "Reading", "Procrastinating"];
   ```

2. Modify `about.handlebars` so that it can accept a value called `{{hobby}}` and display it somewhere useful. For example:

   ```
   <p>I like {{hobby}} on the weekend.</p>
   ```

3. Modify `app.js` so that just before it renders your about page, it picks a random value from the hobbies array and passes it to the view as data.

   Hint #1: Recall that you can get a random array index using `Math.random() * array.length`

   Hint #2: You can pass a JSON object with the property "hobby" and its associated value of your chosen random hobby, into the `res.render` method. Refer to the provided examples if required.

4. Run your app and navigate to http://localhost:3000/. Refresh the page several times. You should notice that the displayed hobby changes each time.

5. Further modify app.js so that in addition to passing a single random hobby to the view (via the "hobby" property), it also passes the *entire array* into the view (via the "hobbies" property).

6. Use the hobbies array within `about.handlebars` to generate an unordered list `<ul>` of all your hobbies.

   Hint: You can use the `{{#each}} {{/each}}` Handlebars *helper*. See the "Contact" page in the provided example for further details.

7. Further modify `about.handlebars` so that, if the hobbies array is null or empty, the page displays the message "I have no hobbies ☹", instead of the content you developed in steps 2 through 6. Test your code by modifying the array definition in app.js appropriately and re-running your app.

   Note: To avoid errors, you may also wish to modify app.js to supply a placeholder string (e.g. "Doing nothing") as the value for the "hobby" property from step 3, if the hobbies array is empty.

   Hint: You can use the `{{#if}} {{else}} {{/if}}` Handlebars helper. See the "Home" page in the provided examples for further details.