CS355: Lab 19 – MVC

In this lab you will fully implement a Model View Controller web app. **To receive credit for this lab, send me the source code or GitHub link to your lab demonstrating the three steps under "Exercises" by the end of the day on 4/23.**

The source code from the example in today's lab can be found here. Both the starting point and the end point: http://www.cs.sonoma.edu/~haderman/cs355/assignments/lab19.zip

MVC Tutorial

You may want to modify lab19-begin. And then modify your own application.

Index Controller

Controllers in the MVC design pattern, or routes in Node.js, are used to take requests from the user, gather data from the model (in our case a MySQL database), then render a view using the data and send it back to the user.

1. Create a folder called controller in your webapp directory.

2. Create a file in that directory named index.js. This will be used to route all requests that go to '/' or '/helloworld', requests that go to other sub-urls like '/user/' will have their own file.

3. At the top of the index.js file add the following lines to configure this file as a router.

   ```
   var express = require('express');
   var router = express.Router();
   ```

4. Move all your app.get('/') and app.post('/') methods to this file.

5. Change the **app.** part of the method call to **router.** So **app.**get('/') becomes **router.**get('/');

6. Add the following line to the end of the file, which will allow other libraries access to its public functions.

   ```
   module.exports = router;
   ```

7. In app.js add the following lines just below your library imports

   ```
   var routes = require('./controller/index');
   ```

8. In app.js add the following lines just before app.listen()

   app.use('/', routes);

9. Browse to http://yourwebsite:yourport/ and verify the home page is routed correctly.

User Controller

1. Create a file in that directory named user.js.

2. At the top of the user.js file add the following lines to configure this file as a router.

   var express = require('express');
   var router = express.Router();

3. Move all the app.get('/user') and app.post('/user') methods to this file. i.e. app.get('/user/create')

4. Change the **app.** part of the method call to **router. AND remove the '/user' prefix from all calls.** So app.get('/user/create') becomes router.get('/create');

5. Add the following line to the end of the file, which will allow other libraries access to its public functions.

   module.exports = router;

6. In app.js add the following lines just below your library imports

   var user = require('./controller/user');

7. In app.js add the following lines just before app.listen()

   app.use('/user', user);

8. Browse to http://yourwebsite:yourport/user/create and verify the webpage is routed correctly.

Models

So now the pages are routing correctly, but the calls to the database no longer work. Let's fix that.

1. Create a folder called models in your webapp directory.

2. Create a file named db.js in the models directory.

3. Add the following line to the top of the db.js file
```
var mysql      = require('mysql');
```

4. Move the Configuration and Create Database sections of app.js to db.js.
```
/* DATABASE CONFIGURATION */
var connection = mysql.createConnection({
      host: 'cwolf.cs.sonoma.edu',
      user: '',
      password: ''
});

var dbToUse = 'ExampleDB'
var createDatabaseQry = 'CREATE DATABASE IF NOT EXISTS ' + dbToUse;
connection.query(createDatabaseQry, function (err) {
      if (err) throw err;

      //use the database for any queries run
      var useDatabaseQry = 'USE ‘ + dbToUse;

      //create the User table if it does not exist
      connection.query(useDatabaseQry, function (err) {
            if (err) throw err;

            var createTableQry = 'CREATE TABLE IF NOT EXISTS User('
                  + 'UserID INT AUTO_INCREMENT PRIMARY KEY'
                  + ',Email VARCHAR(256)'
                  + ',Password VARCHAR(50)'
                  + ')';
            connection.query(createTableQry, function (err) {
                  if (err) throw err;
            });
      });
});
```

5. Move the database query code from out of the router.get() and router.post() methods and create new functions in db.js that handle it. Then modify the functions in user.js to call the database query instead. In the db.js you need to wrap the original database query in a function, and then use “exports.” to make the function call public.

   For instance in db.js add

```
exports.GetAll = function(callback) {
    connection.query('select UserID, Email from User',
        function (err, result) {
            if(err) {
                console.log(err);
                callback(true);
                return;
            }
            callback(false, result);
        }
    );
}
```

And

```
router.get('/user/all', function (req, res) {
    connection.query('select UserID, Email from User',
        function (err, result) {
            res.render('displayUserTable.ejs', {rs: result});
        }
    );
});
```

Becomes

```
router.get('/all', function (req, res) {
    db.GetAll(function (err, result) {
            if (err) throw err;
            res.render('displayUserTable.ejs', {rs: result});
        }
    );
});
```

6.
7. Browse to http://yourwebsite:yourport/user/all and verify that the webpage displays correctly.

Exercises

Modify your project, or another webapp besides the User example, to do the following.

1. The homepage request is routed to an index.js file instead of being handled in the app.js file.

2. A second webpage in a suburl, like /user/create, is routed to a file other than index.js or app.js. i.e. user.js

3. The second webpage submits data to the web app and the function that handles the request passes the data to a database module instead of handling the call to the database directly.