Assignment 2.3 – API Gateway Part II

Instructions

Complete the following:

1. Install express-generator using the NPM manager
    a. npm install express-generator -g
    b. Note: make sure the terminal, command window, or PowerShell window is "Running as an administrator" or "sudo user." If you skip this step, you will have to manually add the npm managers installation directory to your machines environment variable. NPM is typically installed under c:\Users\[your.user.name]\AppData\Roaming\npm
2. Verify the express-generator installation
    a. Enter: express --help
3. Create a new project
    a. Under WEB 420's main directly
        i. Enter: express --view=ejs api-gateway
4. Navigate to api-gateway
    a. cd api-gateway
    b. Run: npm install
5. Install nodemon
    a. Run: npm install nodemon --save-dev
        i. nodemon is a live tracking/editing tool that reloads your Node server when changes are made in the application.
6. Instruct Node to use nodemon when starting your server
    a. Under "scripts" add a new entry
        i. "devstart": "nodemon ./bin/www"
7. Remove /routes/user.js
8. Open the app.js file
    a. Remove the references to users
        i. var users = require('./routes/users');
        ii. app.use('/users', users);
9. Create the following folders in the solutions base directory
    a. controllers
    b. models
10. Add a new JavaScript file to your solutions main directory and name it config.js
    a. This file will serve as a global hub for application level configurations
    b. Add

    Example:

```
var config = {};
config.web = {};
config.web.port = process.env.PORT || '3000';
module.exports = config;
```

11. Replace the hardcoded port value in the bin/www/ file with our config.js files setting
    a. bin/www

    Example:

```
var config = require('../config');
replace the hard coded port with the config.js files port
var port = normalizePort(config.web.port);
```

12. Sign in to mLab and create a new database
    a. Name your new database api-gateway
13. Create a new api-gateway user
    a. admin/admin
14. Configure the MongoDB connection

a. app.js
   i. Remove: var users = require('./routes/users');
   ii. Remove: app.use('/users', users);

Example:

```
var mongoose = require('mongoose');
mongoose.Promise = require('bluebird');

/**
*
Database connection
*/
mongoose.connect('mongodb://admin:admin@ds121588.mlab.com:21588/mean-library', {
        promiseLibrary: require('bluebird')
        }).then ( () => console.log('connection successful'))
        .catch( (err) => console.error(err));
```

15. Build, run, and test the server
    a. Windows
       i. Enter: SET DEBUG=api-gateway:*
       ii. Enter: npm run devstart
    b. Mac OS X and Linux
       i. Enter: DEBUG=api-gateway:*
       ii. Enter: npm run devstart
16. Open a web browser and navigate to localhost:3000
17. Create a User model and schema with three fields, username, password, and email
    a. models/user.js

Example:

```
/**
Fields username, password, and email
*/
var mongoose = require('mongoose');
var userSchema = new mongoose.Schema({ username: String,
password: String, email: String
});
module.exports = mongoose.model('User', userSchema);
/**
Database queries
*/
```

18. Add a public key to the config.js file
    a. config.web.secret = ‘topsecret’
       i. Hopefully it goes without saying, but in a real production application you would not use a plain text string key. Instead, you would set an environment variable
19. Add a new file to the controller’s directory and name it authController.js
20. Add the following lines of code

```
var User = require('../models/user');

// Register a new user on POST exports.user_register = function(req, res) {
res.send('NOT IMPLEMENTED: User registration POST');
};
// Verify token on GET exports.user_token = function(req, res) {
res.send('NOT IMPLEMENTED: User token lookup GET');
```

```
};
```

21. Add a new file to the routes directory and name it api-catalog
22. Add the following lines of code

```
/**
* API Routes
*/
var express = require('express'); var router = express.Router();
var auth_controller = require('../controllers/authController');
// POST request for registering a user router.post('/auth/register',
auth_controller.user_register);
// GET request for verifying user tokens router.get('/auth/token',
auth_controller.user_token);
module.exports = router;
```

23. Add a require statement to the app.js file for the api-catalog routes
    a. var apiCatalog = require(‘./routes/api-catalog’);
24. Register the API Catalog’s routes
    a. App.use(‘/api’, apiCatalog);
25. Build, run, and test the application
26. Using cURL, create the following requests
    a. curl -X GET localhost:3000/api/auth/register
        i. This request should fail
    b. curl -X POST localhost:3000/api/auth/register
    c. curl -X GET localhost:3000/api/auth/token