Assignment Two

Purpose

The purpose of this assignment is to start working with Node.js and become more familiar with HTTP requests.

You will create a standard Node.js server to handle incoming HTTP requests and then respond with information about the request. The server should be able to read incoming header and query parameters and include this information in the response. The server should also only accept certain incoming requests and block others.

You may use any of the standard packages included in Node.js or any third-party packages to complete this assignment.

Requirements

- Create a Node.js server that accepts requests and creates responses and host it on Heroku.
    - The server should accept GET, POST, PUT and DELETE requests. – Any other requests coming in should be rejected.
- Create an environment variable **UNIQUE_KEY** and set it to a value
    - Create .env file for your local development (this file should be .gitignored and not stored in your repository)
    - Create the environment variable on Heroku for your app
- If the server accepts a request, it should respond with information about the request (in a JSON object)
    - The server should respond with the name and value of any **query parameters** sent in. If no headers or query parameters are sent in, then the response should say so.
    - The server should respond with the **environment** variable process.env.UNIQUE_KEY
    - Some headers are generated automatically, like host and user-agent. It's fine to have these appear in the response.
- Include a Basic Auth strategy and JWT strategy
    - For this assignment it is fine to hardcode the username/password within the Auth module you create
- The server should have three different routes that only accept a given HTTP methods, while reject the other methods.
    - /signup
        - HTTP Method: POST
        - If no username or password sent you should return an error
        - If success should return `{success: true, msg: 'Successful created new user.'}`
        - All other methods should return error (e.g. GET, PUT, DELETE, PATCH) - it should respond with a simple statement saying it doesn't support the HTTP method.
    - /signin
        - HTTP Method: POST
        - If user not found you should return an error (401)
        - If success should return `{success: true, token: 'JWT ' + token}`
        - All other methods should return error (e.g. GET, PUT, DELETE, PATCH) - it should respond with a simple statement saying it doesn't support the HTTP method.
    - /movies
        - HTTP Method: GET should return { status: 200, message: 'GET movies", headers: headers: **header from request,** query: **query string from request,** env: **your unique key** }
        - HTTP Method: POST should return {"status": 200, message: "movie saved", headers: headers: **header from request,** query: **query string from request**, env: **your unique key** }

        - HTTP Method: PUT should return {“status: 200, message: “movie updated”, headers: headers: **header from request,** query: **query string from request** , env: **your unique key** }
            - PUT should require authentication (JWT Auth)
        - HTTP Method: DELETE should return {“status: 200, message: “movie deleted”, headers: headers: **header from request,** query: **query string from request**, env: **your unique key** }
            - Delete should require authentication (Basic Auth)
        - All other methods should return error (e.g. PATCH) - it should respond with a simple statement saying it doesn’t support the HTTP method.
    - Any requests made to the base URL (no URN specified) should also be rejected.
- Include a Postman project that can shows all the requirements have been met.
    - This project should include valid requests, as well as requests that fail
    - You should have tests with BasicAuth set to the correct username/password and sets with wrong password
    - You should have tests that signin and retrieve the JWT token that is then used to call the PUT method on /movies – hint (create a signin request and store the token in the environment. Then using that token call the PUT method on /movies

```
var data = JSON.parse(responseBody);
postman.setEnvironmentVariable("token", data.token);
```

or add a pre-req script similar to the following →

```
pm.sendRequest({
      url:  pm.environment.get("OAUTH_URL")+"/uaa/oauth/token",
      method: 'POST',
      header: {
        'Accept': 'application/json',
        'Content-Type': 'application/x-www-form-urlencoded',
        'Authorization': 'Basic Abcdefghijk=='
      },
      body: {
          mode: 'urlencoded',
          urlencoded: [
            {key: "grant_type", value: "password", disabled: false},
            {key: "username", value: pm.environment.get("OAUTH_USERNAME"), disabled: false},
            {key: "password", value: pm.environment.get("OAUTH_PASSWORD"), disabled: false}
        ]
      }
  }, function (err, res) {
        pm.globals.set("token", res.json().access_token);
  });
```

Resources

http://nodejs.org
http://www.passportjs.org/docs/basic-digest/
https://devcenter.heroku.com/articles/heroku-cli
https://devcenter.heroku.com/articles/config-vars
https://devcenter.heroku.com/articles/getting-started-with-nodejs#introduction