Assignment 6

Recommended readings:

- Lecture slides as starting literature
- MDN Links within slides for details
- https://nodejs.org/en/
- http://toolsqa.com/postman-tutorial/
- https://node-postgres.com/

Note: All exercises must be solved using node.js, JavaScript and CSS not using additional libraries or frameworks (except the ones proposed).

Exercise 1 – Basic Concepts

a) Explain the relation between JavaScript and Node.js. Why is it a golden rule of Node.js that of not blocking the event queue?
b) What is Express? What does it offer and what is it used for?
c) Explain the concept of Route in Express. What is the purpose of `app.all()`?

Exercise 2 – First Node.js Gallery

Write a server application using Node.js and Express with the following functionalities:
When */galleryJSON* is requested on your Node.js server, you reply with a JSON file using the proper JSON header. The file should have the same structure as the one for assignment 5.4. However, you now need to read the file gallery.csv, and output the data in the proper JSON output format. You should create corresponding JavaScript objects, and serialize them to JSON for generating the JSON output. Be sure to set the proper content type header.
To test your implementation, create an html document client.html, which calls client.js (see attached file). Make sure you create appropriate html elements with the ids referenced in client.js to display the data. Also make sure that the port specified in client.js (3000) is the same you configured in your node.js installation, otherwise change it accordingly.

Exercise 3 – Setting up the DB connection for a Node.js Gallery

You are now going to set up a Node.js multiuser gallery which interacts with a database for retrieving the images, image information, and users information. Users can interact with the gallery through a gallery API.

Setup the enclosed Node.js gallery API:

- Install Postman, an easy to use Desktop application for conveniently issuing HTTP requests and testing the gallery API (Hint: You don't need to register to be able to use it).
- Place `nodejs_gallery_server` into a convenient location on your computer.
- Open a new console window and navigate to `nodejs_gallery_server`.
- Run '`npm install`' for installing all required Node.js modules.
- Run '`npm install pg`' for installing the Postgres Node.js module if needed.

Open `nodejs_gallery_server` in a text editor/IDE of your choice and take a look at `server.js`:

a) Explain and reason about the structure of this Node.js server application, as well considering the files in the routes folder.
b) Which modules are used and what are their purposes?
c) What is Cross-Origin Resource Sharing (CORS)? What is its purpose?

Setup encloses a PostgreSQL database:

- Install Postgres if you don't have it already.
- Start Postgres and import `db_import/galleryDB.sql` via pgAdmin or the console for creating and populating a new database '`webtech19gallery`'.
- Complete the module '`db.js`': finish '`initDb`' by connecting to the database resolving/rejecting the Promise on success/failure. The database should be accessible by any server component, simply by requiring and calling the '`getDb`' function (e.g. see '`gallery.js`').
- Test your implementation by opening a console window, navigating to `nodejs_gallery` and running '`npm start`'. Expected output:

```
Database is connected ... Listening
on port 3000...
```

Exercise 4 – Familiarizing with the Node.js Gallery API

a) Leave the server running, open up Postman and issue a simple GET request to '`localhost:3000`'. Explain the output.

b) What changes would you need to make in order to pass and output a simple parameter to the API (e.g. calling '`localhost:3000/hello`')? Change the server's default route ("/") response to also returning such a passed parameter, e.g. "`Welcome to gallery server 1.0 - you passed the parameter hello.`" How can you additionally set a response status using express? Hint: Always kill (Ctrl+C) and restart the server using '`npm start`' for changes to take effect when altering the server code.

Exercise 5 – Implementing the Node.js Gallery API

You are now going to implement a first functioning API for the image gallery in its several parts, so that its functionalities can be exposed to the users.

a) Naïve login: complete the module '`login.js`', so that it queries the database, checking whether the password provided in the parameter '`:pass`' is correct for the user identified by the parameter '`:email`'. Valid email/password combinations can be determined by inspecting the database of '`galleryDB.sql`' (Hint: For the sake of simplicity, the database stores passwords in plain text – never do this for serious projects, use salted hashing!). Upon success/failure return appropriate response status codes depending on the query outcome:
   - `400`: an error occurred
   - `401`: login failed
   - `200`: login successful

   A successfully logged in client should also receive a JSON containing the first and last name of the user as a response.
   Test and debug your implementation with a POST request in Postman such as:
   `localhost:3000/login/sandy@nomail.com/`*`12345`*.

b) Complete the module '`gallery.js`', which should implement retrieving a user specific gallery as list of image identifiers: users can own the same or different images (defined by table '`users_images`') and this route should return all image ids for the user whose email address is passed as parameter.
   The gallery should be returned as list of image ids in the following JSON format:
   `{[Id1, …, Idn]}`.
   Test and debug your implementation using GET requests in Postman such as:

`localhost:3000/gallery/sandy@nomail.com`.

c) Extend the module '`gallery.js`', so that it implements retrieving a specific image for a specific user gallery given the image id.
The image should be returned like in the following JSON format:
```
{"dataSmall": "img/beach_small.jpg",
"dataBig": "img/beach.jpg",
"description": "Goleta Beach near UCSB Campus"}.
```
Test and debug your implementation using GET requests in Postman such as:
`localhost:3000/gallery/sandy@nomail.com/1`.

d) Provide users with the functionality to change image descriptions by implementing the code in the module '`image.js`'. You should pass image id and description using the body ('`req.body.description`') in JSON format (Hint: When using body and JSON, you must include '`"Content-Type": "application/json"`' within the headers). Test and debug your implementation using PUT requests in Postman.