SÃO PAULO UNIVERSITY
DEPARTMENT OF COMPUTER SCIENCE
SCC0219 - Introduction to Web Development

Nome: Michelle Wingter da Silva
nUSP: 10783243

Exercises 6: Node.js

1. **What is the Event Loop and how is it used by Node.js? (2 pts)**

O Event Loop é o que permite ao Node.js executar operações de E/S, apesar de o JavaScript ser single-threaded (ou seja, que uma aplicação executa apenas uma thread por vez), descarregando as operações para o kernel do sistema sempre que possível. O Event Loop fica em execução esperando novos eventos para tratar, e, para cada requisição, um novo evento é criado.

2. **Create a simple Key-value store that saves JSON values indexed by string keys using Node.js (these strings do not have blanks or special characters). It will have just 3 operations: read, write, and delete:**
   a. **Read: returns the value associated with a key. If the key is not valid, it returns null.**
   b. **Write: receives a key-value pair (both in JSON) and write them in the database.**
   c. **Delete: receives a key and delete the key-value pair that it refers to.**

   These 3 operations should use the same route: `/store`. You should separate them using the appropriate HTTP method for each operation (GET, PUT, POST, or DELETE). Any necessary parameter must be included using the appropriate technique (ex: query strings). For instance, a GET request for the value of key `john` would be:

   ```
   http://localhost:8080/store/john        OR
   http://localhost:8080/store?query=john
   ```

   The key-value pairs can be recorded in a variable on the server using a Javascript object (remember that all Javascript objects are maps). There is no need to use a database or save these variables to disk. There is no authentication. Any client can read, write, or delete.

   Tips:
   1. **Follow the video class up to lesson 9 and you will end with a running server.**
   2. **In lesson 10, it explains the REST actions (GET, PUT, POST, and DELETE) and shows how to test the requests using the Postman tool (install in lesson 3).**
   3. **After lesson 10, you have a running server that has GET, PUT, and DELETE services.**
   4. **Which services do you need for your system? The server already has operations for these 3 services. For the PUT, it receives the id, but for the GET, it does not. Use the PUT router as a model and modify the GET router to receive an id too. Test.**
   5. **Right now, the server just sends back the status information. Modify the router implementations to store and send back the information you need. Remember**

that you can only receive and send back text to the client. Use `JSON.parse()` and `JSON.stringify()` functions to convert to and from text whenever needed.
(4 pts)

3. **The file `userinfo.html` has a vue.js application that reads information about users, using the JSON format, from a mocked server (a server that simulates web services). To run this app, put it in an Http server (in Linux: `python3 -m http.server 8000`). It works but only implements the read (GET) functionality. Modify this application to use the Key-value store (from the last question) to implement the read(), create(), and delete() functions (only these 3 functions need to be modified/created). For the 3 functions, use the username input as the key.**
   (4 pts)

4. **Challenge: Create a user registration system using a Node.js backend and a Vue.js client. This system must have:**
   a. **A login vue.js page with:**
      ○ **A login panel with username and password fields and a login button. If the user supplies a correct username/password, it opens the main page. If not, the application will popup an error message.**
      ○ **A panel to create new users. In it, users must provide a username, password, email, and address to create new system users. Optionally, that panel can be implemented as another page, and, in this case, the login page will have a button to jump to it. After creating a new user, the system returns to the login page.**
   b. **The main vue.js page with:**
      ○ **The user information: username, password, email, and address.**
      ○ **A button to delete the user. When a user is deleted, a popup message is shown and the system returns to the login page.**
      ○ **A button to return to the login page.**

   Both vue.js pages can be implemented using basic components or, if you want, you can use the login interface you are writing for the Pet Shop. The code created here can be reused in the PetShop project. For security reasons, the server (not the client) should check:
   - **If the username has no spaces or upper case characters and if it is greater than 5 characters and smaller than 9.**
   - **If the email is properly formatted (`<something> @ <something>`).**
   - **If the password is, at least, bigger than 8 characters and has no spaces.**
   - **If the address has, at least, 10 characters.**

   In case of errors, an error message will popup (stating the kind of error) and no new user will be created. There is no need to delete the problematic field(s), the user can do that.
   Before coding, think about which services the server has to have. Give them URLs and decide which arguments they will need and which information they will send back.
   Notice:
   - **There is no need to create an actual user section, using cookies or other technologies.**

- **Data can be recorded in variables on the server. There is no need to use a database or save these variables to disk.**
- **To communicate data between client and server, you may use any data format, but an easy option is to use JSON (look for `JSON.parse()` and `JSON.stringify()` functions).**

Tips:

- **Begin implementing the server.**
    a. **Follow the video class up to lesson 9 and you will end with a running server.**
    b. **In lesson 10, it explains the REST actions (GET, PUT, POST, and DELETE) and shows how to test the requests using the Postman tool (install in lesson 3).**
    c. **After lesson 10, you have a running server that has GET, PUT, and DELETE services.**
    d. **Which services do you need for your system? Read user info, write user info, and delete user info. The server already has operations for these 3 services. For the PUT, it receives the id, but for the GET, it does not. Use the PUT router as a model and modify the GET router to receive an id too. Test.**
    e. **Right now, the server just sends back the status information. Modify the router implementations to store and send back the information you need. Remember that you can only receive and send back text to the client. Use `JSON.parse()` and `JSON.stringify()` functions to convert to and from text whenever needed.**
- **For the client, use the Forms app you created in Exercise 5.**
    a. **Modify it to read the information you need.**
    b. **You will need two pages. You may use the Vue.js Router (Vue video lesson 32 or/and code example), but can also use div tags to change what content is being shown on the page (Ex: `<div style="visibility: hidden">`.**
    c. **To communicate with the server, use `this.$http.get` (.get, .post, .put) as shown at Vue video lesson 18.**
- **Implement the format checks (for username, email, etc) and the error messages in the server and client.**

(+2 pts)

Good Work!