**Introduction**

To program, use the “strict” mode of JavaScript, the indentation and comments.

Use local URLs in your files, so your production works regardless of the URL of the server.

In all your functions that respond to HTTP requests, put a **try {} catch {}** to catch all the exceptions, and show the error messages.

Test your production by launching node in your main folder, not in the server or client folder. If you are reading a “db.json” file in the “server” folder, open it by reference to the path “server/db.json” (not “db.json” or “../server/db.json”).

**Use of NodeJS modules**

NodeJS is based on the concept of **modules**. A module is a block of JavaScript code that you can load and use in your code, that is to say as a library. To load a module, NodeJS uses the **require** function. It is used like this:

**var module = require ('module\_name');**

There are many modules available for NodeJS. Some are installed automatically with NodeJS (see the documentation [Here](https://nodejs.org/api/)). In this case, you just have to use **require** to use the module. This is the case for the **fs** module of management of the “File System”, used to read / write files (see the documentation of this module [here](https://nodejs.org/api/fs.html)), or even for the module **http** which allows to receive and send HTTP messages (see the documentation of this module [here](https://nodejs.org/api/http.html)).

NodeJS provides the command line tool **npm** to load more modules. To download and install a module, use the following command line:

**$ npm install module\_name**

You can verify that a module is installed by checking that there is a subdirectory with the module name in the **node\_modules** folder. You can then use **require**.

For example, to install the **express** module which is an http server, use:

**$ npm install express --save**

And then, in the JavaScript code, write:

**const express = require('express')**

**const app = express()**

**app.get('/', (req, res) => res.send('Hello World!'))**

**app.listen(3000, () => console.log('Example app listening on port 3000!'))**

**REST API**

You will create a REST API on the JSON database. A REST-compatible API, or “RESTful”, is a programming interface application that uses HTTP requests to get (GET), place (PUT), publish (POST) and delete (DELETE) data.

Please use the file name: **server2.js** for your production.

**Question 0:** Create a server that just says **Hi**, listening to a port number given on the command line. Then add simple logging of requests. You can use the module **morgan** for logging.

The command line will then be, to start the server on port 8000: **node server2.js 8000**

To get the command line arguments use the array **process.argv**

Modify your server to respond to:

* **http://localhost:8000/exit** to make the server stop and exit,
* **http://localhost:8000/restore** which simply reloads db.json in memory, and answers in plain text “db.json reloaded”.

The two are necessary.

**Question 1:** Create a server that, to the request **http://localhost:8000/countpapers**, replies the number of publications documented in db.json. The answer is sent in plain text.

**Question 2:** Add the processing of the request GET with URL **http://localhost:8000/authoredby/xxx**, which answers the number of publications where the name of one of the authors contains xxx, ignoring the case of letters. This answer is plain text.

After doing it simply by getting the **xxx** from the **request.url** field, note that express has a feature of naming fields in the URL, with prefix : See the doc [here](https://expressjs.com/en/guide/routing.html)

Then use that feature to simplify your code.

**Question 3:** Add the processing of the request GET with URL **http://localhost:8000/descriptors/xxx**, which answers the descriptors of publications whose names of authors contain **xxx**, ignoring the case of letters. This answer is in JSON, so should have the media type **application/json**

Note: the descriptor of a publication is the object describing the publication, with all its properties. A descriptor is one of the objects in the array in db.json.

Example:

**{**

**"key": "8000304",**

**"title": "Assessment of Fetal Exposure to 4G LTE Tablet in Realistic Scenarios: Effect of Position, Gestational Age and Frequency",**

**"journal": "IEEE Journal of Electromagnetics, RF and Microwaves in Medicine and Biology",**

**"year": "2017",**

**"month": "aug",**

**"keywords": "Fetus Dosimetry, Specific absorption rate, Radio frequency, Pregnancy, Electromagnetics, Antennas Author Keywords Radio frequency, Dosimetry, Polynomial chaos, Pregnancy",**

**"lang": "en",**

**"authors": [**

**"E. Chiaramello",**

**"M. Parazzini",**

**"S. Fiocchi",**

**"P. Ravazzani",**

**"j. wiart"**

**],**

**"category": "article",**

**"state": "published",**

**"dept": "comelec",**

**"group": "rfm"**

**}**

**Question 4:** Add the processing of the request GET with URL **http://localhost:8000/titles/xxx**, which answers the titles of publications whose names of authors contain **xxx**, ignoring the case of letters. This answer is in JSON, so should have the media type **application/json**

**Question 5:** Add the processing of the request GET with URL **http://localhost:8000/publication/xxx**, which responds the descriptor of the publication whose “key” is **xxx**.

**Question 6:** Add the processing of the request DELETE with URL **http://localhost:8000/publication/xxx**, which deletes the publication whose “key” is **xxx** in the database that is in memory. No need to save the database for this exercise. Check the deletion by querying the base with countpapers, author, or titles.

To send a DELETE query, you can use the command line: **curl -X DELETE http://localhost:8000/publication/xxx**

**Question 7:** Add the processing of the request POST with URL **http://localhost:8000/publication**, which adds an **imaginary** publication to the database that is in memory. No need to save the database for this exercise. Check the addition by querying the base with publication, countpapers, authoredby, or titles.

Please use **imaginary** as the key of this new publication.

A POST request is basically sending a multi-part mime structure with your data in it.

In order to process the upload of the new reference, you should use the express middleware **express.json** which will hide from you the complexity of receiving the multipart mime and extracting the data from it.

To send the request, create a file called **createpost.txt** in which you will write the **curl** command that posts your new reference.

One question is: how to get the POST-ed data from the system. This is where the above-mentioned **express.json** middleware comes handy.

If you define **app.use(express.json());**

Then, you can use the field **request.body** to get the JSON data that was posted.

**Question 8:** Add the PUT request **http://localhost:8000/publication/xxx**, which changes the publication whose the “key” is **xxx** in the database that is in memory. No need to save the database for this exercise. Check the addition by querying the base with countpapers, authoredby, or titles.

A PUT request is basically sending a multi-part MIME structure with your data in it, like POST.

In the PUT JSON data, just add the fields that you are modifying. Fields already present and not in PUT data will be kept as they are.

To send the request, create a file called **createput.txt** in which you will write the **curl** command that puts your modified reference.