**Lab: Web Application**

The purpose of this lab is to study the concepts discussed during [INF203 on Web server technologies](http://perso.telecom-paris.fr/dufourd/cours/inf203/).

Rather than install a turnkey web server (such as Apache), set up (e.g. modify httpd.conf configuration files) and add specific behaviors in languages like PHP, Java, C / C ++ …, this lab proposes to create a server from scratch (or almost). You will program in JavaScript, using NodeJS and its support for HTTP / HTTPS protocols.

**To do**

For this lab, you will need to upload the different JavaScript files you created. Everything will have to be zipped and dropped below.

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

***Thank you for using zip (and not tar and gzip or bzip2). Zip all files directly into a single zip, do not zip the folder.***

***No spaces or accented characters or special characters in the name of the zip file.***

***This lab is to be done alone***.

Please respect the file names and IDs we ask you to use, otherwise automatic grading will not work and you will not have the grade matching your work.

**Use local URLs in your files**, so your production works regardless of the URL of the server, and so that it also works on the grading machine.

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

To debug any problem, please make sure you have opened the JavaScript console of your browser and looked for error messages.

In Chrome, “Inspect” in the context menu (right click), option-cmd-C on a Mac, ctrl-shift-I on Windows.

In Firefox, “Inspect Element” in the context menu (right click), option-cmd-C on a Mac, ctrl-shift-C on Windows.

In Safari, “Inspect Element” in the context menu (right click), option-cmd-C on a Mac.

In Edge, “Inspect Element” in the context menu (right click), F12 Developer Tool on a Windows PC.

**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 init //only do this line the first time

> npm install module\_name --save

npm init creates a package.json. You can use the default response to most questions.

The --save option adds a dependency to the package.json file which is required for the grader to know that it needs to load this extra module.

If you have a problem with npm, for example it does not exist on your system, look [there for instructions](https://labgrader.r2.enst.fr/files/webapp/webappLab.html#npm)

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 http module which is a simple http server, use:

> npm install http

And then, in the JavaScript code, write:

var http = require('http');

**Additional useful information**

Additional useful information can be found in: [Labs Overview](https://perso.telecom-paristech.fr/dufourd/cours/inf203/labs-overview.html)

**Exercise - A pseudo data base**

Reuse the server.mjs you developed in the Simple Server Lab, renamed to server1.mjs to create / edit a database and then to display the contents of the database.

The pseudo database will be a text file containing JSON data. Put the client in a client folder. The JSON file will be storage.json. The storage.json file will have the structure:[{"title": "foo", "color": "red", "value": 20}, {"title": "bar", "color": "ivory", "value": 100}, ...]

Values are strictly positive numbers. Add values to a total that will represent 100% and 360° of the pie. Display percentages on each wedge.

**Question 0:** Take the code of the simple server lab and remove all but:

* answer something to URL “/” to help you make sure that the server works, possibly with a name and version number
* serve files from the current execution folder when the URL starts with “/Root/”
* exit the server when receiving the URL “/exit”, this just to help me with keeping the grading server simple. Thank you.

Make sure your storage.json contains at least two slices.

**Questions 1 to 5:** Client-side: create a button bar with “show txt”, “add element”, “remove element”, “clear” and “restore”. The actions of these buttons are:

1. show txt: shows the text of the current JSON
2. add element: adds in the current JSON an element whose information is a number (value), a text (title) and a color in CSS format (color name or hex) (color). The new element is at the end of the list.
3. remove element: deletes an element in the running with his index finger
4. clear: deletes all elements in the current JSON, and replaces the whole storage.json content with [{"title": "empty", "color": "red", "value": 1}]
5. restore: restores a storage.json file with 3 slices, this last action is intended to simplify the (semi) automatic grading.

Put the buttons at the top of the space as a menu bar and the display space below. Each button clears the display space and shows what is necessary. For each button, there is something to do in the editing page and something to do in the server.

In order to allow auto-grading, please put ids on the buttons in the toolbar:

* “SHOW\_BUTTON” for the button to show the text of the current JSON below the menu bar,
* “BUT\_ADD” for the button to show the form to add a new element to the JSON,
* “REMOVE” for the button to show the form to remove an existing element from the JSON,
* “CLEAR” for the button to clear the existing JSON,

You also need textfields for the index to remove and the title, color and value to add, use “indexTF”, “titleTF”, “colorTF” and “valueTF” for them. The buttons to validate addition and removal shall have “DOADD” and “VALIDREM” as ids. The element in which you insert the result of show shall have the id “MAINSHOW”.

Server: Make sure that the JSON on the server is changed. URLs querying the server will be:

1. http://localhost:8000/Items called by the button show txt
2. http://localhost:8000/add?title=\*&value=\*\*&color=\*\*\* where \* is a string, \*\* a number and \*\*\* a CSS color, called by the button add element
3. http://localhost:8000/remove?index=\* where \* is a number, called by the button remove element
4. http://localhost:8000/clear called by the button clear
5. http://localhost:8000/restore called by the button restore

File name: client/test2.html,client/test2.js, server1.mjs