Liaison avec un serveur nodeJs

Prerequisite

- JRE Java
- nodeJS
- npm

Architecture

There is no Kalima Javascript Node, however there is a JS API to link a JS program (a nodeJS server for example) with a specific Java Node (KalimaNodeAdmin).

In this tutorial we will see how to develop a nodeJS server that uses the Kalima JS API, then how to set and run the Java part and the nodeJS server.

Finally the nodeJS server will be tested with a simple curl request. You can refer to the REST API documentation to test other requests:
https://doc.kalimadb.com/APIs/Rest/restAPI.htm.

NodeJS

First, create a new directory for your project.

Librairie Kalima

In your projet directory, you need to copy the Kalima JS API. Create a directory named «libs » for example, and copy the kalima directory that you can find in etc/lib/js.

app.js

Then create a JS file, which will be the entry point of the server, in our case, app.js:

```
require("dotenv").config();

var express = require("express");
var bodyParser = require("body-parser");
fs = require("fs");
var kalimaApiRouter = require("./routes/kalimaApiRouter");
var app = express();

app.use(bodyParser.json());
app.use(express.json());
app.use(express.urlencoded({ extended: false }));

app.listen(process.env.PORT);
console.log(`app listening on port ${process.env.PORT}`);

app.use("/api", kalimaApiRouter);

module.exports = app;
```

As we can see, we call kalimaApiRouter, which we will create right after.

In addition, this code call an environment file wich we will set later.

kalimaApiRouter.js

Then you need to add the router (kalimaApiRouter.js) which will use our Kalima JS API. For this, you can create a directory named “routes in your project”, and add a new file named kalimaApiRouter.js:

```
var express = require("express");
var router = express.Router();
var kalimaApi = require("../libs/kalima/kalimaApi");
var kalimaEventsApi = require("../libs/kalima/kalimaEventsApi");

kalimaApi.init(process.env.FILES_PATH);
kalimaEventsApi.init(process.env.FILES_PATH);

router.get("/events", function (req, res) {
  kalimaEventsApi.addClient(req, res);
});

router.get("/*", function (req, res) {
  kalimaApi.get(req, res);
});

router.delete("/*", function (req, res) {
  kalimaApi.delete(req, res);
});

router.post("/*", function (req, res) {
  kalimaApi.post(req, res);
});

module.exports = router;
```

Note that this example remains as simple as possible and doesn't contain any security level. You are free to add authentication for example.

Install dependencies

This code depend on few API that you can install with npm:

```
npm install dotenv
npm install express
npm install btoa
```

Configurations

Java

As we describe before, our nodeJS server will communicate with a Java Node: KalimaNodeAdmin. Like other Java Node, you need to run it with one parameter: The path of a config file.

So create a new config file, node.config for example:

```
SERVER_PORT=9100
FILES_PATH=/home/rcs/jit/KalimaNodeJS/
# CHANGE IT WITH THE SERIALID GRANTED AFTER ADMINISTRATIVE VALIDATION
SerialId=KalimaNodeAdmin
PRIVACHAIN=org.kalima.tuto
```

- SERVER_PORT ➔ Choose a free port
- FILES_PATH ➔ You can choose any directory you want. This directory will contain few mandatory files for the application, and the logs of the Java Node
- SerialId ➔ This ID allow your node to be authorized on the Kalima Blockchain (if you used the inscription form for tutorials, you have 10 serial id that you received by mail)
- PRIVACHAIN ➔ wich Allows you to choose the blockchain on which the node will be connected, for tutorials: org.kalima.tuto.

The KalimaNodeAdmin.jar is in etc/lib.

nodeJS

Our nodeJS server will also read a config file. So create a new .env file:

```
PORT=9000
FILES_PATH=/home/rcs/jit/KalimaNodeJS/
```

- PORT ➔ Listening port of your nodeJS server
- FILES_PATH ➔ Must be the same to that of the JAVA part

Run

Start by running the java part:

```
Java -jar KalimaNodeAdmin.jar node.config
```

Then in an another console, run the nodeJS server:

```
node app.js
```

In a third console, test the application with a curl request:

```
curl http://localhost:9000/api/cache/list
```

In return, you must obtain a JSON containing the list of addresses on which your Java node is authorized.