



Integração de Sistemas

2016 / 2017

Work 2

REST Web Services

Monitoring Cloud Platform

Duration: 4 classes, helped by teacher

Delivery: 7th May

1 Introduction

Web Services importance in the communication and integration of systems has been growing at an exponential rate in recent years. There are three mostly widely used Web Services: SOAP, WSDL and REST. Despite referred as such, neither is a Web Service per say. Whereas SOAP refers to a XML-based protocol that exchanges information between applications, WSDL is a XML document that describes a Web Service. On the other hand, REST is simply an architecture for networked systems which uses standards such as HTTP. As integration tool, one of the best features of Web Services is of not specifying the way how data is fed to or generated by them. Modern day applications are mostly based on REST Web Services due to their reliability and speed. Some applications such as Socket.io and NodeJS take advantage of these type of Web Services for instantaneous and seamless integration. This work pretends to explore new technologies that make the most of REST Web Services for obtaining/inserting/etc. data in and out of a database in a fast and seamless way to the user.

2 The Problem

With the constant growth of the Internet of Things, with devices becoming more and more capable of communicating over a network, along with the improved capacities of the Internet itself, it is becoming simpler than ever to monitor appliances of our day to day usage. This monitoring ranges from energy consumption to water consumption, walking through usage analysis. It is expected that users are capable of starting and stopping monitoring a device. The data received from the devices should then be stored and made available for consulting by an external device. Lastly, an external device (browser, app) should be able to consult this information and display it to the user. Moreover, it should be possible to assign and remove devices to and from users.

2.1 Work description

In order to provide these features, it is expected that the students develop an infrastructure (Figure 1), accessible through the Internet, that allows a user to register new devices, remove old ones and monitor existing ones. It is also expected that a backend capable of providing the user with these features exists.

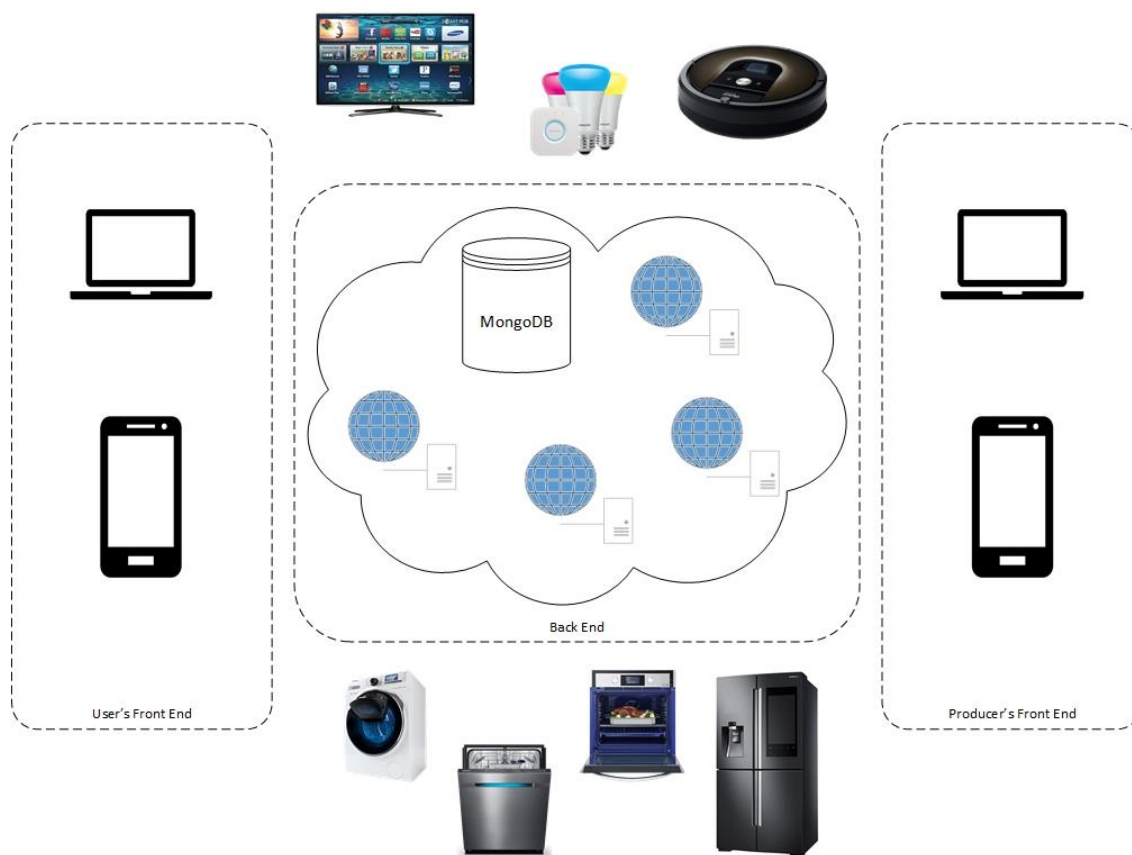


Figure 1 – Infrastructure to implement.

That being said, the front end should enable users to manipulate their devices and monitor them. Producers can also use the front end to, e.g., add new models of devices. This involves data manipulation in and out of a database, i.e., storing and querying data from a database along with monitorization in real-time. That being said, this work involves four phases of data manipulation: generation, storage, processing and visualization. The devices will generate data, that can be obtained, processed and presented in real-time. This data will then be stored in a database for later access. The generated data can also be processed before or after being stored in the database (real-time or historical analysis, respectively). Lastly, the data can be visualized in the form of, e.g., graphs.

There is a number of minimum features that should be made available by the Front End:

- Common:
 - Register themselves in the platform (sign in system);
 - Login in the system;
 - Logout from the system;
- Users should be able to:
 - Add a new device to be monitored (register a new device in the platform);
 - Stop a device from being monitored (this does NOT unregister the device);
 - Unregister a device (unregister a device from the platform);
 - Consult devices history.
- Producers should be able to:
 - Add/remove product models;
 - Add/remove device types;
 - Consult history of a given device;
 - Consult history of a given user;
 - Consult history of a given device type.

2.2 Database

The database will keep track of the users and devices registered in the platform. It also stores all the data generated from the devices. In order to keep track of all the information in the platform, the component model in Figure 2 will be used.

The database to be used is a NoSQL one. This means the concept of relationships between the data does not exist. However, the data is, of course, correlated between each other. That being said, the main object in the database is the Device, which belongs to a given ProductModel, is of a certain DeviceType and is in a certain state. It may also occur in some status of SystemError. Moreover, it generates usages logs. The Device is produced by the Producer and used by the User.

There will be three possible states for the devices, previously defined in the database, in the “states” collection:

- **ON** – when set to this state, the device is turned on, working and being monitored;
- **OFF** – when set to this state, the device is turned off;
- **UNMONITORED** – when set to this state, the device is turned on but is not being monitored.

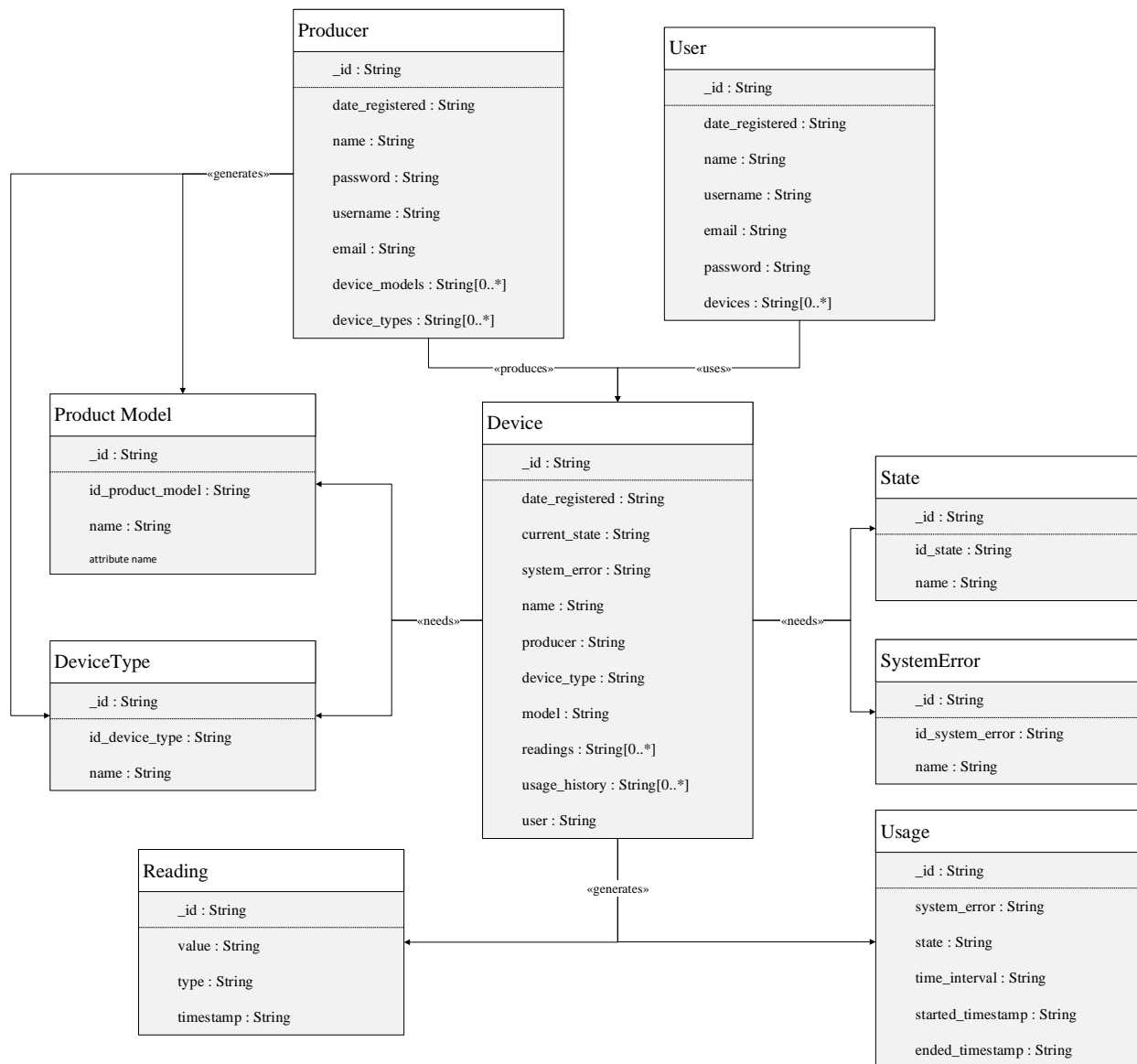


Figure 2 – Database component model.

There are times, however, when something may go wrong. To emulate such events, there are also some pre-defined errors in the database, namely:

- **NO_DATABASE_CONNECTION** – the device enters this state when it loses the connection with the database;
- **NO_ERROR** – no error found;
- **NO_DATA_EXTRACTION** – the device is not being capable of extracting data;
- **UNRECOGNIZED** – when the device throws an error that is none of the above;

There is no more data in the database at the beginning besides the one already mentioned. Further data that is stored there should be made available for future interactions with the database.

3 Implementation

For the implementation of this work, you will be using the following:

- **NPM** – the package manager for JavaScript. Allows you to find and reuse other developers code;
- **Socket.io** – fastest and reliable real-time engine.
- **Express.js** – web framework for Node.js, an asynchronous event driven JavaScript runtime;
- **Mongoose.js** – framework for MongoDB object modelling for Node.js;
- Text editor (**Atom** is recommended);
- **MongoDB** – database for storing data;
- **Java and JavaScript** – programming languages for this work.
- **Windows 10/Linux**;
- **Code** supplied by the teacher.

4 Planning

You were provided with two separate files: “DeviceEmulator.zip” and “Monitoring System.zip”. The first contains the code provided by the teacher for the device emulator project, which emulates the behaviour of real life devices. The latter contains the code provided by the teacher for the network infrastructure (front end, back end and database). Each of these files should be extracted into two separate folders with the same name as the compacted files.

In the folder “Monitoring System”, given to you by the teacher, you can find three folders:

- **backend** – where all the methods for the backend of the platform will be developed. This was developed using NodeJS, more specifically ExpressJS, and all the methods should be developed using this technology;
- **frontend** – where all the methods for the frontend of the platform will be developed. A first demo version of this was developed using ReactJS. You can use whatever you want to develop the frontend (what is inside this folder is just a demonstration);
- **database** – where the database of the platform will be stored. Interfacing with the database will be handling using MongooseJS library.

In the folder “Device Emulator” you will find a Java Project that simply starts a REST server that is capable of answering requests for adding, removing and changing the state of a device. It is

expected, and desired, that you alter the code given to you by the teacher in order to expand the possibilities of the platform.

To help you understand how should all the projects interlink among each other please see Figure 3. This figure illustrates the sequence that should be followed by the platform in order to create a new device. The remaining operations should follow, in general, the same structure despite some obvious changes.

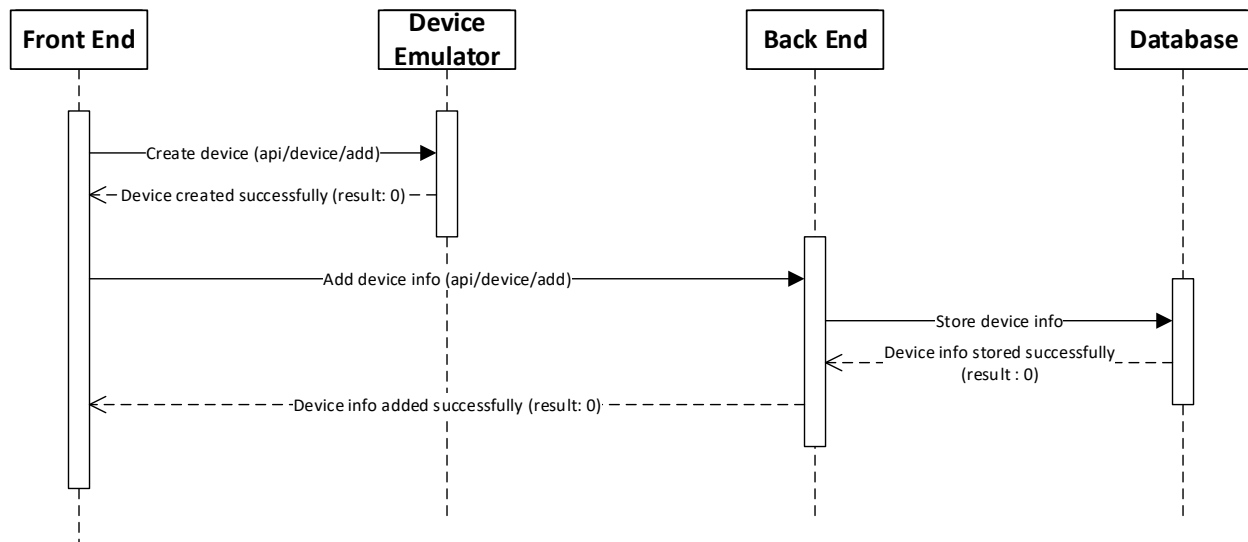
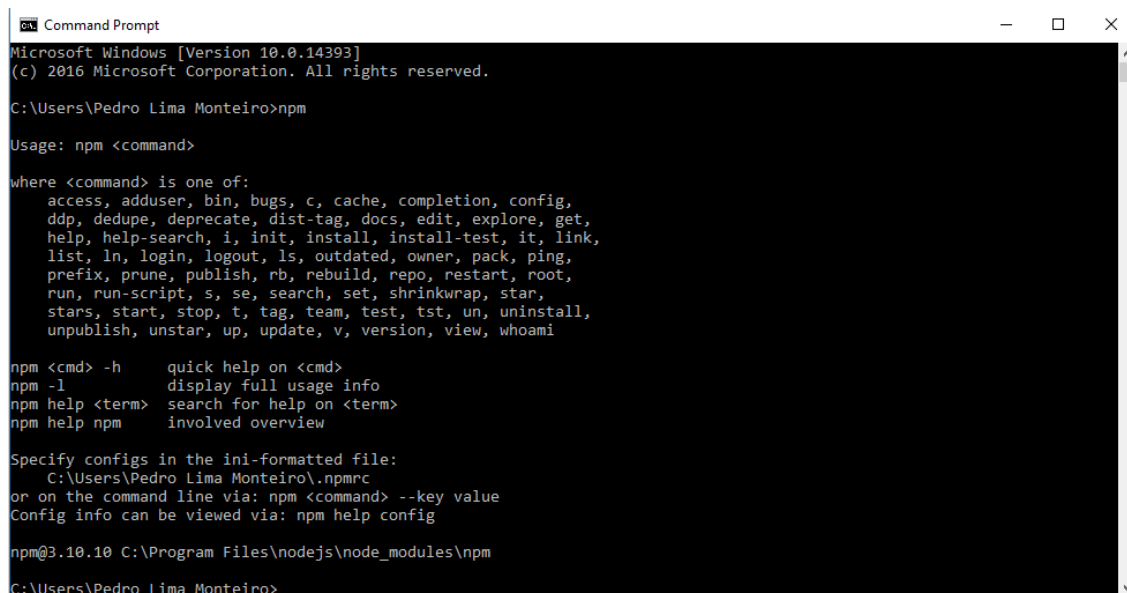


Figure 3 – Device creation sequence diagram.

4.1 Lesson 1 – Installation and Tutorials

Note: Whenever prompted for it, allow for access. It is also assumed that you have Java SDK and Netbeans installed on your computer.

- 1) Install Node.js (also installs NPM);
 - a. Go to this [link](#) and download the appropriate version for your OS;
 - b. Launch the setup and follow instructions until the end;
 - c. Start your command line and you should be able to introduce the “npm” command and get the response in Figure 4.



```
Command Prompt
Microsoft Windows [Version 10.0.14393]
(c) 2016 Microsoft Corporation. All rights reserved.

C:\Users\Pedro Lima Monteiro>npm

Usage: npm <command>

where <command> is one of:
  access, adduser, bin, bugs, c, cache, completion, config,
  ddp, dedupe, deprecate, dist-tag, docs, edit, explore, get,
  help, help-search, i, init, install, install-test, it, link,
  list, ln, login, logout, ls, outdated, owner, pack, ping,
  prefix, prune, publish, rb, rebuild, repo, restart, root,
  run, run-script, s, se, search, set, shrinkwrap, star,
  stars, start, stop, t, tag, team, test, tst, un, uninstall,
  unpublish, unstar, up, update, v, version, view, whoami

npm <cmd> -h      quick help on <cmd>
npm -l           display full usage info
npm help <term>  search for help on <term>
npm help npm     involved overview

Specify configs in the ini-formatted file:
  C:\Users\Pedro Lima Monteiro\.npmrc
or on the command line via: npm <command> --key value
Config info can be viewed via: npm help config

npm@3.10.10 C:\Program Files\nodejs\node_modules\npm
C:\Users\Pedro Lima Monteiro>
```

Figure 4 – Node.js and NPM installed successfully.

2) Install MongoDB;

- Go to this [link](#) and download the appropriate version for your OS (use “Windows Server 2008 R2 64-bit and later, with SSL support x64” for Windows 10);
- Launch the setup and follow the instructions (choose Complete when prompted) until the end;
- Go to your “Advanced System Settings” (in Windows 10, right-click “This PC” and choose “Properties”);
- Go to “Environment Variables”, double-click the “Path” variable in the top box, choose “New” and “Browse”. Navigate to where you installed MongoDB and select the “bin” folder. You should end up with something similar to Figure 5;

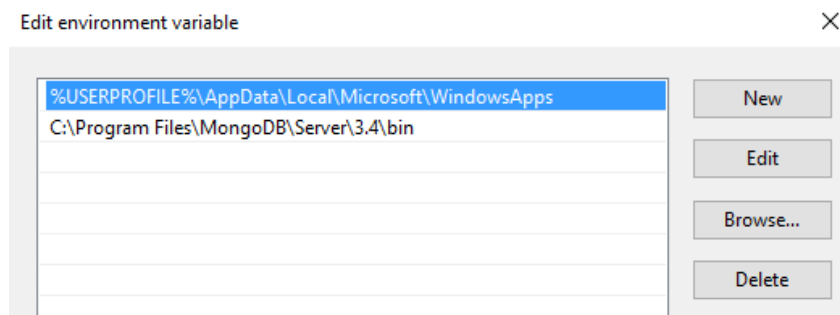


Figure 5 – Environment Variable setup for MongoDB.

- e. Add another Environment Variable, one that holds the path to where your database should be stored. Go to “Environment Variables” as in d) and press “New”. In “Variable name” insert whatever you want, e.g., “db_path” and in “Variable value” use browse directory and select the directory “database” inside the “Monitoring System” folder the teachers gave you. You should end up with something similar to Figure 6;

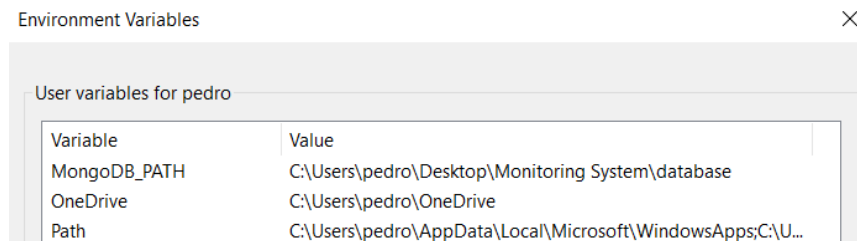


Figure 6 – Database path environment variable setup.

- f. You now should be able to open your command line and type: “mongod --dbpath “%db_path%”” (note that db_path is the name of the environment variable set in e)). This should set the database up and running with an output similar to that of Figure 7;

```
(c) 2016 Microsoft Corporation. All rights reserved.

C:\Users\pedro>mongod --dbpath "%MongoDB_PATH%"
2017-04-03T02:12:51.936+0100 I CONTROL [initandlisten] MongoDB starting : pid=7424 port=27017 dbpath=C:\Users\pedro\Desktop\Monitoring System\database 64-bit host=DESKTOP-0ITNDVC
2017-04-03T02:12:51.936+0100 I CONTROL [initandlisten] targetMinOS: Windows 7/Windows Server 2008 R2
2017-04-03T02:12:51.937+0100 I CONTROL [initandlisten] db version v3.4.3
2017-04-03T02:12:51.937+0100 I CONTROL [initandlisten] git version: f07437fb5a6cca07c10bafa78365456eb1d6d5e1
2017-04-03T02:12:51.937+0100 I CONTROL [initandlisten] OpenSSL version: OpenSSL 1.0.1u-fips 22 Sep 2016
2017-04-03T02:12:51.937+0100 I CONTROL [initandlisten] allocator: tcmalloc
2017-04-03T02:12:51.937+0100 I CONTROL [initandlisten] modules: none
2017-04-03T02:12:51.937+0100 I CONTROL [initandlisten] build environment:
2017-04-03T02:12:51.938+0100 I CONTROL [initandlisten] distmod: 2008plus-ssl
2017-04-03T02:12:51.938+0100 I CONTROL [initandlisten] distarch: x86_64
2017-04-03T02:12:51.938+0100 I CONTROL [initandlisten] target_arch: x86_64
2017-04-03T02:12:51.938+0100 I CONTROL [initandlisten] options: { storage: { dbPath: "C:\Users\pedro\Desktop\Monitoring System\database" } }
2017-04-03T02:12:51.939+0100 I STORAGE [initandlisten] wiredtiger_open config: create,cache_size=7635M,session_max=20000,eviction=(threads_min=4,threads_max=4),config_base=false,statistics=(fast),log=(enabled=true,archive=true,path=journal,compressor=snappy),file_manager=(close_idle_time=100000),checkpoint=(wait=60,log_size=2GB),statistics_log=(wait=0),
2017-04-03T02:12:52.045+0100 I CONTROL [initandlisten] ** WARNING: Access control is not enabled for the database.
2017-04-03T02:12:52.045+0100 I CONTROL [initandlisten] ** Read and write access to data and configuration is unrestricted.
2017-04-03T02:12:52.045+0100 I CONTROL [initandlisten]
2017-04-03T02:12:52.200+0100 I FTDC [initandlisten] Initializing full-time diagnostic data capture with directory 'C:\Users\pedro\Desktop\Monitoring System\database\diagnostic.data'
2017-04-03T02:12:52.231+0100 I INDEX [initandlisten] build index on: admin.system.version properties: { v: 2, key: { version: 1 }, name: "incompatible_with_version_32", ns: "admin.system.version" }
2017-04-03T02:12:52.231+0100 I INDEX [initandlisten] building index using bulk method; build may temporarily use up to 500 megabytes of RAM
2017-04-03T02:12:52.236+0100 I INDEX [initandlisten] build index done. scanned 0 total records. 0 secs
2017-04-03T02:12:52.237+0100 I COMMAND [initandlisten] setting featureCompatibilityVersion to 3.4
2017-04-03T02:12:52.239+0100 I NETWORK [thread1] waiting for connections on port 27017
```

Figure 7 – MongoDB Setup.

- 3) Install Atom (optional);
 - a. Go to this [link](#) and download the appropriate version for your OS;
 - b. Launch the setup and follow the instructions until the end;
 - c. You should be able to launch Atom by now.
 - d. Choose a folder where to put the compact file given to you by the teacher and extract it. You should end up with a “Monitoring System” folder. You can delete the compact file now;
 - e. For Atom’s users, add a project folder for each of the folders inside the compact file sent to you;
 - i. Open Atom;
 - ii. Go to “File”, “Add Project Folder”, navigate to the “backend” folder and add it;
 - iii. Repeat the same process for the “frontend” folder;
 - iv. You should end up with something similar to Figure 8.

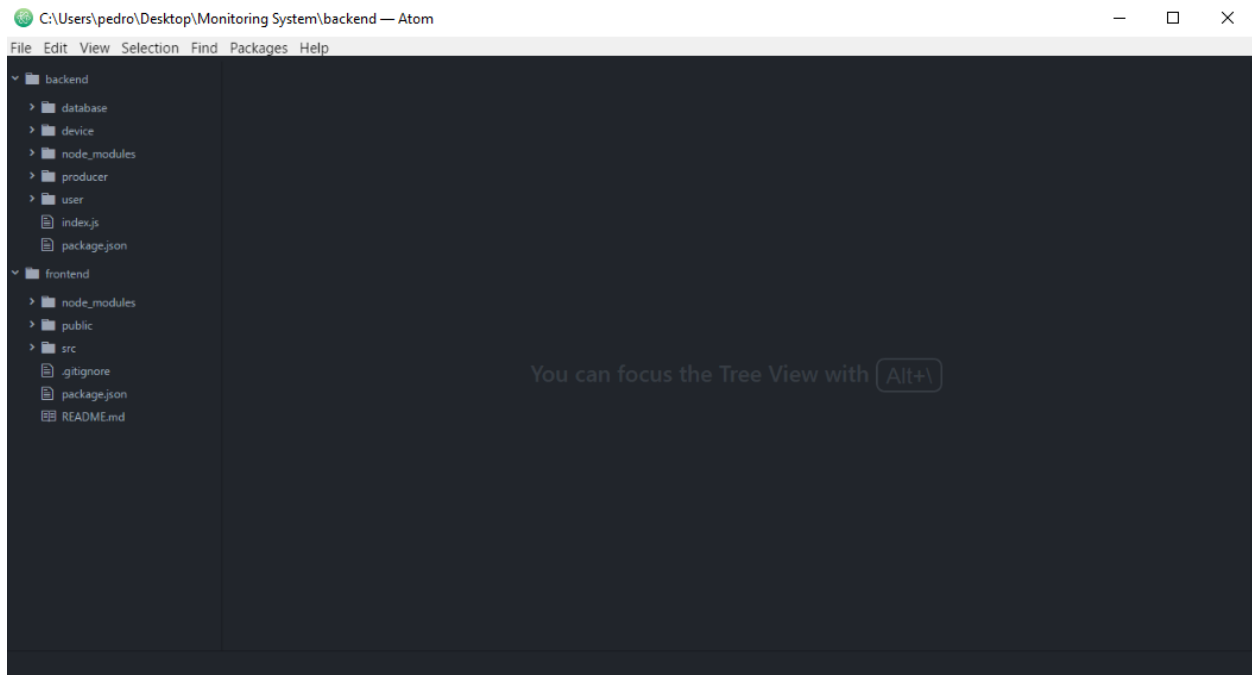
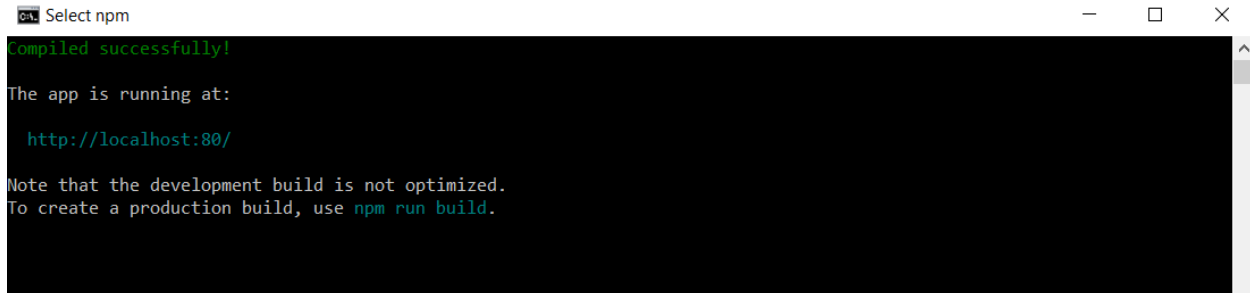


Figure 8 – Atom Setup.

- 4) You should now be able to launch the platform for the first time. In order to do so successfully, one should do it in the following order;
 - a. Launch the Device Emulator project in Netbeans. You may need to “Build with dependencies” given that it is a Maven project. Please note that this project does not record any state of the system when shutdown (this can be altered by you if you

want to). This means that devices added and stored in the database upon initializing the back end and its connection to the database will not be present in the Device Emulator instance if it has been shut down meanwhile (this can be changes as an Extra functionality).

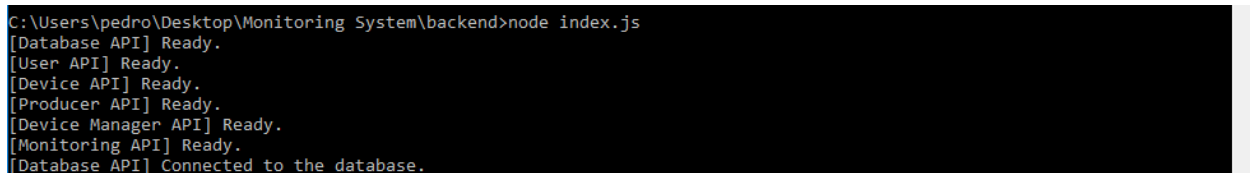
- b. Launch the frontend module of the platform (it does not really matter if this is launched before or after the remaining modules of the platform);
 - i. To do so, open a command line and navigate to “Monitoring System/frontend”;
 - ii. Type “npm start” and it should start launching the frontend application and you should end up with something similar to Figure 9. A new browser window should also have opened automatically.



```
CL Select npm
Compiled successfully!
The app is running at:
  http://localhost:80/
Note that the development build is not optimized.
To create a production build, use npm run build.
```

Figure 9 – Frontend successful start.

- c. Launch the database by proceeding as in 2.f). This must be done before launching the backend module as it will not start before the database is launched;
- d. Launch the backend module of the platform;
 - i. To do so, open a command line and navigate to “Monitoring System/backend”;
 - ii. Type “node index.js” and it should start launching the backend application and you should end up with something similar to Figure 10.



```
C:\Users\pedro\Desktop\Monitoring System\backend>node index.js
[Database API] Ready.
[User API] Ready.
[Device API] Ready.
[Producer API] Ready.
[Device Manager API] Ready.
[Monitoring API] Ready.
[Database API] Connected to the database.
```

Figure 10 – Backend successful start.

- 5) Now that you have successfully setup and launched the platform for the first time, take some time to go through the code provided by the teacher and try to understand what was developed as an infrastructure for your code;
 - a. Walkthroughs for Javascript;

- i. Basics;
 - ii. ES6.
- b. Walkthroughs for Socket.io;
 - i. Basics;
 - ii. Socket.io for Java.
- c. Walkthroughs for NodeJS/ExpressJS;
 - i. Basics for NodeJS;
 - ii. Basics for ExpressJS;
 - iii. ExpressJS API.
- d. Walkthroughs for MongooseJS;
 - i. MongooseJS Homepage;
 - ii. MongooseJS Essentials.
- e. Extra tutorials;
 - i. Encryption;
 - ii. Design;
 - iii. Tokens;

4.2 Lesson 2 – Producer and User Methods

In this lesson, you should be able to develop the methods (both frontend and backend) that allows a producer and a user to perform basic operations in the platform (see Section 2.1).

4.3 Lesson 3 – Monitoring Methods

In this lesson, you should be able to develop the methods that allow you to monitor a device. This includes the methods for generating data in the Device Emulator project and presenting them in the front end.

4.4 Lesson 4 – Login and Extra functionalities

In this lesson, you should be able to develop the methods (both frontend and backend) that enable both users and producers to login into the platform (do not forget to secure the platform against unsecure access). Moreover, you should take the extra time to think about and create extra functionalities for the platform (try to think out of the box!).

5 Evaluation

The evaluation of this work will go as follows:

- Correct implementation of the minimum required methods for the producer and the user (see Section 2.1);
 - Maximum 7 classification score.



- Correct implementation of the minimum required methods for monitoring devices;
 - Maximum 3 classification score.
- Extra functionalities implementation for user and producer;
 - Maximum 2 classification score.
- Extra functionalities implementation for monitoring devices;
 - Maximum 6 classification score.
- Correct implementation of safety procedures;
 - Maximum 1 classification score.
- Appealing design of the frontend and clean code;
 - Maximum 1 classification score.

6 Delivery

To deliver this work, you should send an e-mail to “pedro.monteiro@uninova.pt” with the subject “[IS] Delivery Work 2” ([IS] Entrega Trabalho 2) with a .zip (or equivalent) file with both folders, “Monitoring System” and “Device Emulator” zipped. The work must be delivered until 23:59 of 7th May. For each hour of delay, your mark will be downgraded by 1.

Teachers

| | |
|---------------------|--|
| Pedro Lima Monteiro | pedro.monteiro@uninova.pt |
| Ricardo Peres | ricardo.peres@uninova.pt |
| José Barata | jab@uninova.pt |