**RemoteMD**

* **Project description**

RemoteMD is a web application interface that helps a remote doctor to communicate with his patient via a robot.

The doctor can keep tracking his patient by moving the robot around the patient's home or by receiving updates from the robot.

* **Installation - NodeJS:**

1. Windows:
   1. Download NodeJS installer: <https://nodejs.org/en/download/>
   2. Run the installer and follow instructions.
2. Linux (Ubuntu):
   1. 'sudo apt-get install nodejs'
   2. 'sudo apt-get install npm'
3. Ensure NodeJS has been installed: run 'node -v' in your cmd.
4. Update version of npm: 'npm install npm --global'
5. Create package.json file: 'npm init' and follow instructions.
6. Install packages: 'npm install <package name>'
   1. in Robot: express, ws, spawn-handler, readline.
   2. in Cloud: express, http, ws, fs, body-parser, mongoose, cookie-parser, express-session.
7. Install MongoDB:
   1. Download from <https://www.mongodb.org/> and install
   2. Create directory: <path to dir>/RobotMD/db
   3. Run: mongod --dbpath "<path to dir>/RemoteMD/db"

* **Running the project**

1. The Robot: in the folder contains robot\_app.js file run   
   'node robot\_app.js'
2. The cloud (website): in the folder contains cloud\_app.js file run   
   'node cloud\_app.js'

* **Files description**

1. views folder:
   1. index.html & index.css – contains the homepage content: login page.
   2. doctor.html & doctor.css – the doctor's page: choosing a Robot.
   3. navigation.html & navigation.css – navigation page: contains navigation control buttons and the Robot's state window.
   4. script.js – JavaScript code for navigation page.
   5. Icons folder – icons images.
2. routes folder:
   1. index.js – manages routing.
3. cloud\_app.js – manages a web server: handles user's requests and manages log files.
4. robot\_app.js – manages the Robot's server.

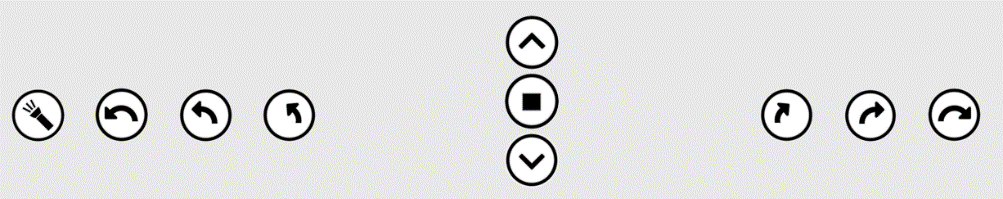
* **How does it work?**

The connection between the browser and the cloud, and between the cloud and the robot established by Websockets.  
The cloud and the Robot are **Websocket servers –** simply a TCP application listening on any port.  
In out case, the cloud listening on port 80 and the Robot on port 8080.  
The browser and the cloud are **Websocket clients –** application that use the websocket API to communicate with websocket server.

Further reading: <https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API>

* **The Robot**

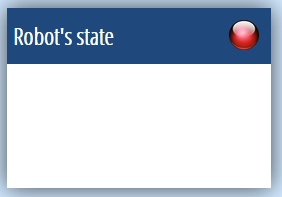
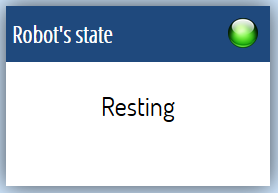
The Robot consists of a Arduino board and a LIDAR.   
 Arduino and LIDAR code runs by a C++ program named MAPING.exe, and our Robot's server runs MAPING.exe by a NodeJS module (named spawn).  
through this module we are passing commands to the program and passing back indications (confirmation/success/failure).

* **Commands and Responses**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Commands** | | **Responses** | | |
| **description** | **Representation** | **Confirmation** | **Success** | **failure** |
| Flashlight ON/OFF | FL\_1 / FL\_0 | C\_FL\_1 / C\_FL\_0 | S\_FL\_1 / S\_FL\_0 | F\_FL\_1 / F\_FL\_0 |
| Turning <num> degrees left.  Num: {20, 45, 90} | TU\_-<num> | C\_TU\_-<num> | S\_TU\_-<num> | F\_TU\_-<num> |
| Moving forward (30 cm) | FO\_30 | C\_FO\_30 | S\_FO\_30 | F\_FO\_30 |
| Stop | ST | C\_ST | S\_ST | F\_ST |
| Moving backwards (30 cm) | FO\_-30 | C\_FO\_-30 | S\_FO\_-30 | F\_FO\_-30 |
| Turning <num> degrees right.  Num: {20, 45, 90} | TU\_<num> | C\_TU\_<num> | S\_TU\_<num> | F\_TU\_<num> |
| Move to coordinates <x>, <y> | MV\n<X>\n<Y> | C\_MV\_<X>\_<Y> | S\_MV\_<X>\_<Y> | F\_MV\_<X>\_<Y> |
| Get map | MAP | C\_MAP | OBS|map size:<X>x<Y>|<list of obstacles – eg 111,222 456,789> | F\_MAP |

* The Robot sending response in JSON form: { conf: <msg> }
* Responses appears in the Robot's state window.
* **Robot's state**





* The Robot has 3 states: connected, closed and error.
* The cloud informs the browser by sending JSON: { robot\_conn: <state> }