

Intelligent Security System connected to IoT

Frederic Lopes Goncalves Magalhaes

Bachelor's Thesis

Degree Program in BIT

2017



Author(s) Frederic Lopes Goncalves Magalhaes	
Degree program Business Information Technology	
Report/thesis title Intelligent Security System connected to IoT	Number of pages and appendix pages 35 + 1
<p>A home is a private place which means safe and secure. Everyone would like to keep an eye on his place but unfortunately doing it physically 24/7 is not possible. Fortunately, the technologies available nowadays can be used to create an automated third eye for those who want to keep a regular control on their home.</p> <p>Mobile Application has become common in many business sectors and is a trend as more and more people possess a smartphone. Then, with two major mobile application stores, Google Play for Android devices and Apple store for iOS devices, developing a mobile application is a way to reach millions of customers.</p> <p>This essay will be about developing and implementing an intelligent security interface for an indoor environment on a low consumption computer in order to control connected objects and gather motion events built on the latest web technologies.</p> <p>The thesis is a Front-End developed as a hybrid mobile application based on Apache Cordova for a fast development for multiple platforms and a Back-End as a MEAN stack application (MongoDB, Express.js, Angular.js, Node.js).</p> <p>All product is developed using JavaScript and CSS3 frameworks such as Ionic framework for the mobile application user interface, Angular.js for dynamic client-side and Express.js for Node.js as web server framework.</p> <p>The testing phase is conduct with a functional test of all system working together and a feasibility test.</p> <p>It is concluded with a discussion about the different technologies used and the researches carried out through the product development.</p>	
Keywords Internet of Things (IoT), Motion, Security, Mobile	

Table of contents

1	Introduction	1
2	Research Question.....	2
2.1	Objective of the project	2
2.2	Scope of the project	2
3	Background technologies	3
3.1	Back-End Side	3
3.2	Front-End Side	3
4	Application Architecture and Design	4
4.1	Application Architecture	4
4.1.1	Back-End Side	4
4.1.2	Front-End Side	5
4.2	Application Design	6
4.2.1	Database design.....	6
4.2.2	Components Diagram	6
4.2.3	Class Diagram	8
4.2.4	Sequence Diagram – User Authentication.....	9
5	Implementation	10
5.1	Pre-requirements	10
5.2	FFMPEG with x264.....	10
5.3	Motion for Raspberry Pi	11
5.4	Security	12
5.4.1	Hash.....	13
5.4.2	Helmet.....	14
5.4.3	HTTPS	14
5.5	Internet of Things	17
5.5.1	Required components	17
5.5.2	Electronic diagrams	18
5.5.3	GPIOs controlled over Python.....	19

5.6	Ionic Application User Interface.....	20
6	Testing.....	23
6.1	JSON Web Token.....	24
6.2	Internet of Things.....	25
6.3	HTTPS.....	27
7	Discussion.....	30
7.1	MEAN stack.....	30
7.2	JWT.....	30
7.3	Motion.....	32
7.4	Internet of Things.....	33
7.5	An introduction to electronic.....	33
7.5.1	Electronic symbols.....	33
7.5.2	Ohm's law.....	34
7.5.3	Series and parallel circuits.....	34
7.6	Conclusion.....	35
	References.....	36
	Appendices.....	37

Abbreviations

AJAX	Asynchronous JAVascript & XML
BSON	Binary JSON
CRUD	Create Read Update Delete
CSS	Cascading Style Sheets
DB	Database
DBMS	Database Management System
FFMPEG	Fast Forward Moving Picture Experts Group
GPIO	General Purpose Input/Output
HTTP	HyperText Transfer Protocole
HTTPS	HyperText Transfer Protocole Secure
IoT	Internet of Things
JOSE	Javascript Object Signing & Encryption
JPEG	Joint Photographic Experts Group
JSON	JavaScript Object Notation
JWE	JSON Web Encryption
JWS	JSON Web Signature
LED	Ligth-Emitting Diode
MPEG	Moving Picture Experts Group
MVC	Model View Controller
NOSQL	Not Only Structured Query Language
NPM	Node Package Manager
OS	Operating System
REST	Representational State Transfer
SPA	Single Page Application
SQL	Structured Query Language
SSL	Secure Socket Layer
TLS	Transfer Layer Security
XSS	Cross-Site Scripting

1 Introduction

The Internet of Things (IoT) is increasingly growing as the cost of internet is decreasing throughout the past years and IP version 6 (IPv6) is taking hold. Combined with the low cost of the hardware, the current era is offering a wide opportunity to develop new automated solutions where anybody has the opportunity to create his/her own home-made connected objects.

Furthermore, open source algorithms and technologies are increasing, allowing anybody to expand his/her creativity by combining ready-to-use technologies. Per example, companies such as Google, Arduino, Raspberry Pi Foundation are creating and donating researches to the community allowing them to develop, fix issues and innovate.

Nowadays, IoT and security solutions sold on the market are individual control devices, licensed where data flow cannot be managed by the user himself. Thus, the data is going through a third-party server, which most of the time the user has no information about such as the location and the laws about data coming in and going out of the country where the server is located. Moreover, these solutions are static and cannot be customized, requiring multiple products with multiple software's to use them.

The research result of this thesis will be a fully developed distributed-control system which can be implemented in an indoor environment using the existing electrical wiring for a minimal cost of 60€, controlled from a smartphone or browser and a full control on data flow where the user himself manage it. The gathered data can then be used to do deep learning and create an artificial intelligence able to recognize its owners, understand their behavior to automate indoor objects and ease their life in their indoor environment. (Ansari, Sedky, Sharma, & Tyagil, 2015)

This document will describe the technologies used to develop a product able to control indoor objects, detect motion events and manage all data by storing it in a private database, allowing the user to choose the server location of the product. The product is a back-end side composed of a server, camera, objects, database and a front-end side as a mobile application for having a user interface and managing the product.

Firstly, we will define the objectives and the scope of the project. Then, we will study the background technologies of the project by listing the different technologies used. We will design the product using Unified Modeling Language (UML) in order to understand the product behavior and its architecture. Then we will implement the product on a low consumption computer and on a mobile device. Finally, we will test the product and discuss about the choices made and the results gathered.

2 Research Question

2.1 Objective of the project

The aim of this thesis is to create a real-time home view solution that allows any authenticated user to be informed about physical events in an indoor environment on distant devices and have an approach of IoT by providing an access to connected objects.

The system has to implement an encryption technology in order to prevent outsiders from seeing or manipulating data sent or used by an authenticated user.

This will be an introduction to a future automated security system able to recognize its owners in the indoor environment allowing it to stay on 24/7 and learn their behavior.

2.2 Scope of the project

This project is an all product that informs authenticated users about suspicious events in an indoor environment. It has to be composed of a front-end as a mobile application for multiple devices giving access to multiple features of the mobile phone. The scope for the front-end side is:

- A user can take a picture with his camera and send it to the server as an authentication method.
- An access to real-time connected objects in the indoor environment split by room.
- A list of the different events gathered by the system.
- Read and delete videos or pictures stored on the server side.

The back-end part will be in charge of gathering events, storing data and pushing notifications to authenticated clients through a protected connection between client and server. As communication between both has to be encrypted, the HTTPS has to be implemented in order to prevent outsiders from viewing or modifying the communication.

Then, the scope of the back-end part is:

- A camera able to detect motions and saving them as pictures and videos.
- An API to authenticate a user with his username and password and generate an encoded token as a session.
- An API for getting connected objects in the indoor environment and be able to know or change their state in real-time with sockets.
- An API for getting files stored in the database such as pictures and videos. This API has to be able to stream medias.
- HTTPS has to be implemented to prevent attacks.

3 Background technologies

In order to create the product, we are going to use the following technologies:

3.1 Back-End Side

- Operating system: Raspbian on Raspberry Pi 2, 3 or more powerful (any other computer would be able to run it but for a low energy consumption we decided to implement it on a Nano computer).
- Languages: Express for Node.JS, Python, Bash Command Language.
- Database: NoSQL MongoDB.
- Technologies: HTTPS, Socket.io, Motion, JSON Web Token, Passport.
- Tools: IntelliJ from JetBrains.

3.2 Front-End Side

- Operating system: Hybrid mobile (iOS, Android, Windows Phone, etc...).
- Languages: HTML5, CSS3, JavaScript framework AngularJS.
- Technologies: Ionic framework for Apache Cordova, AngularJS, ngCordova.
- Tools: IntelliJ from JetBrains.

4 Application Architecture and Design

The application will be developed as a full JavaScript solution, named MEAN which is the abbreviation of four used technologies: MongoDB as the database, Express as the web server framework, AngularJS as web client framework and Node.js as the server.

The advantage of using a full-stack JavaScript solution is the centralization of a unique programming language. Plus, it will enforce the use of the Model View Controller (MVC) architecture and the process of serialization and deserialization, which is the conversion and reconversion of data into a format, of data structures as the data marshalling is based on JSON objects in back-end and front-end side. (Haviv, 2014)

4.1 Application Architecture

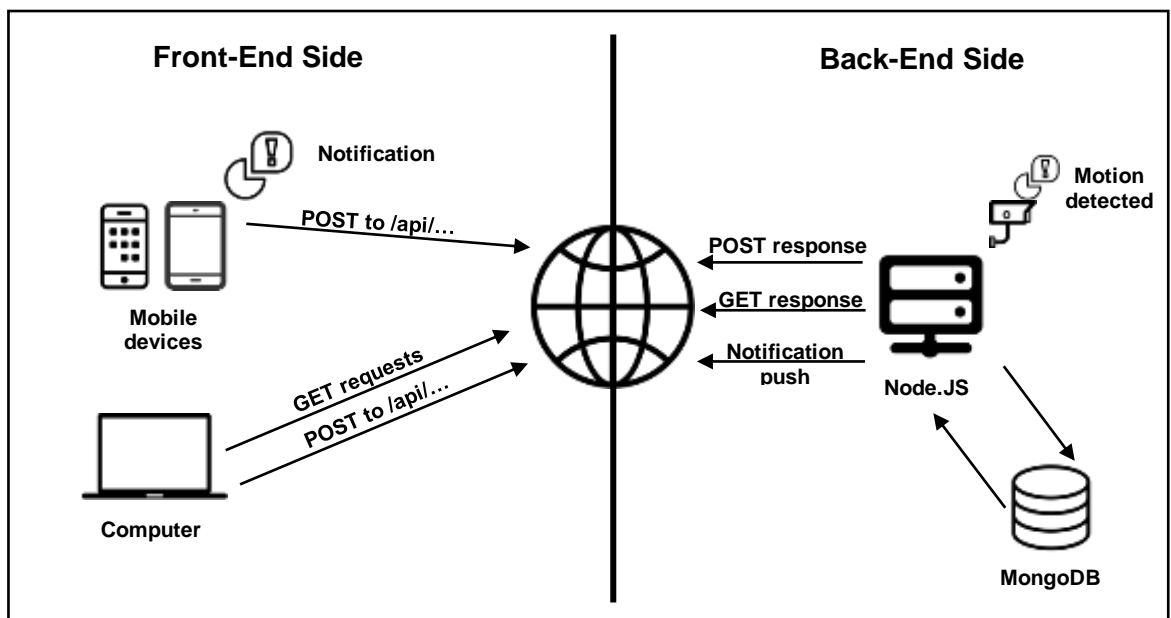


Figure 1. Architecture of the application

Based on Figure 1, the solution will be composed of two sides:

4.1.1 Back-End Side

As we are developing a MEAN solution, the Back-End will be composed the server developed entirely on JavaScript using Express framework for Node.JS. Despite PHP¹ is the most famous server side language to do dynamic web pages, using Node.js is an opportunity to discover a new technology to develop dynamic web applications but also

¹ PHP is an open source programming language for creating dynamic web pages.

take advantage of all community dependencies available on Node Package Manager (NPM).

We will store the data using the Not Only Structured Query Language (NoSQL) MongoDB instead of a traditional DBMS (e.g. MySQL²). Compared to the traditional databases where data is stored in tables, MongoDB is using a Binary JSON (BSON) data model, designed for web applications purpose which stores hierarchical documents in standard formats such as JSON and XML. This choice brings many advantages such as a higher scalability and a fast read operations as the application won't have to rebuild the objects.

4.1.2 Front-End Side

The product has to be presented as a mobile application in order to access mobile devices features (e.g. push notification, camera) as we need to inform each client about a threat but also as a web application for being able to connect through a computer browser.

A solution is to develop a mobile application which is hybrid and time effective. The research conduct to use Ionic framework for Apache Cordova framework, which provides components based on Bootstrap, framework developed by Twitter, and ready to use. Furthermore, Apache Cordova provides access to mobile devices features as you can see on table 1 hereinafter.

Feature	Android	iPhone	Windows Phone	Firefox OS	Ubuntu touch
Camera	YES	YES	YES	YES	YES
Notification (alert, sound, vibration)	YES	YES	YES	YES	YES
Storage	YES	YES	YES	YES	YES
Media	YES	YES	YES	YES	YES
Geolocalisation	YES	YES	YES	YES	YES

Table 1. Cordova supported features (cordova.apache.org/docs/en/latest/guide/support/)

As Apache Cordova framework is based on the latest web technologies (AngularJS, HTML5, CSS3), developing a web application based on the mobile application only requires to customize the HTML and CSS in order to have a browser compatibility.

² MySQL is an open source relational database.

4.2 Application Design

This section will show the different UML diagrams of the product.

4.2.1 Database design

As we are using the NoSQL MongoDB, we are storing hierarchical documents which are often schemaless. Although, we draw the Physical Model Design of our database as it was a traditional relational database.

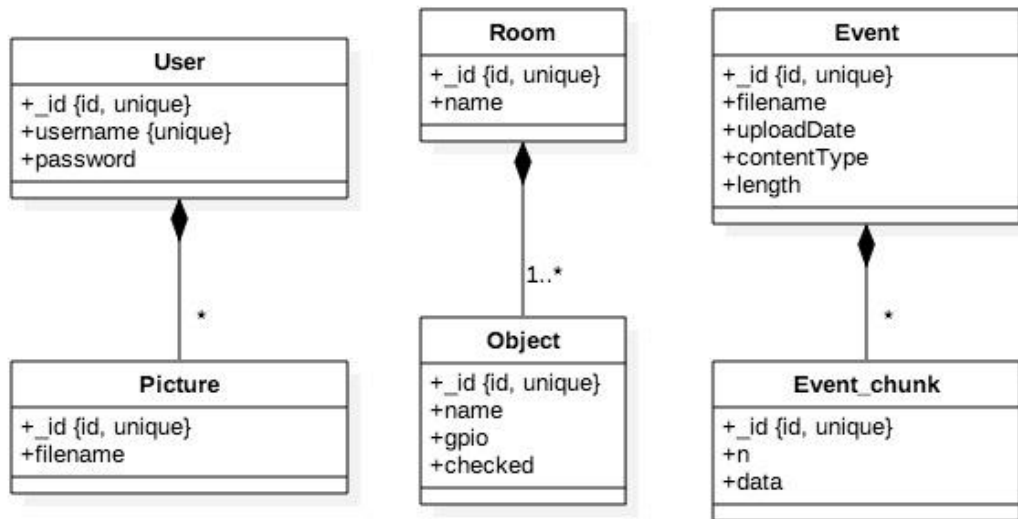


Figure 2. Physical Model Design

Based on figure 2, the database is composed of:

- A User table containing users' credentials for authentication process.
- A Room table in order to sort connected objects represented by Object table by room.
- An Event table containing general information about an occurred event which is split in many parts stored in Event_chunk table as MongoDB is only able to store documents with a size of 16mb.

As said previously, a document-based database will allow us to modify the database structure throughout the development without having to reconsider the current structure.

4.2.2 Components Diagram

The component diagram shows the different components of the product from the server side to the client side and their relations helping to have a better overview of all available interfaces throughout the communication process and their dependencies.

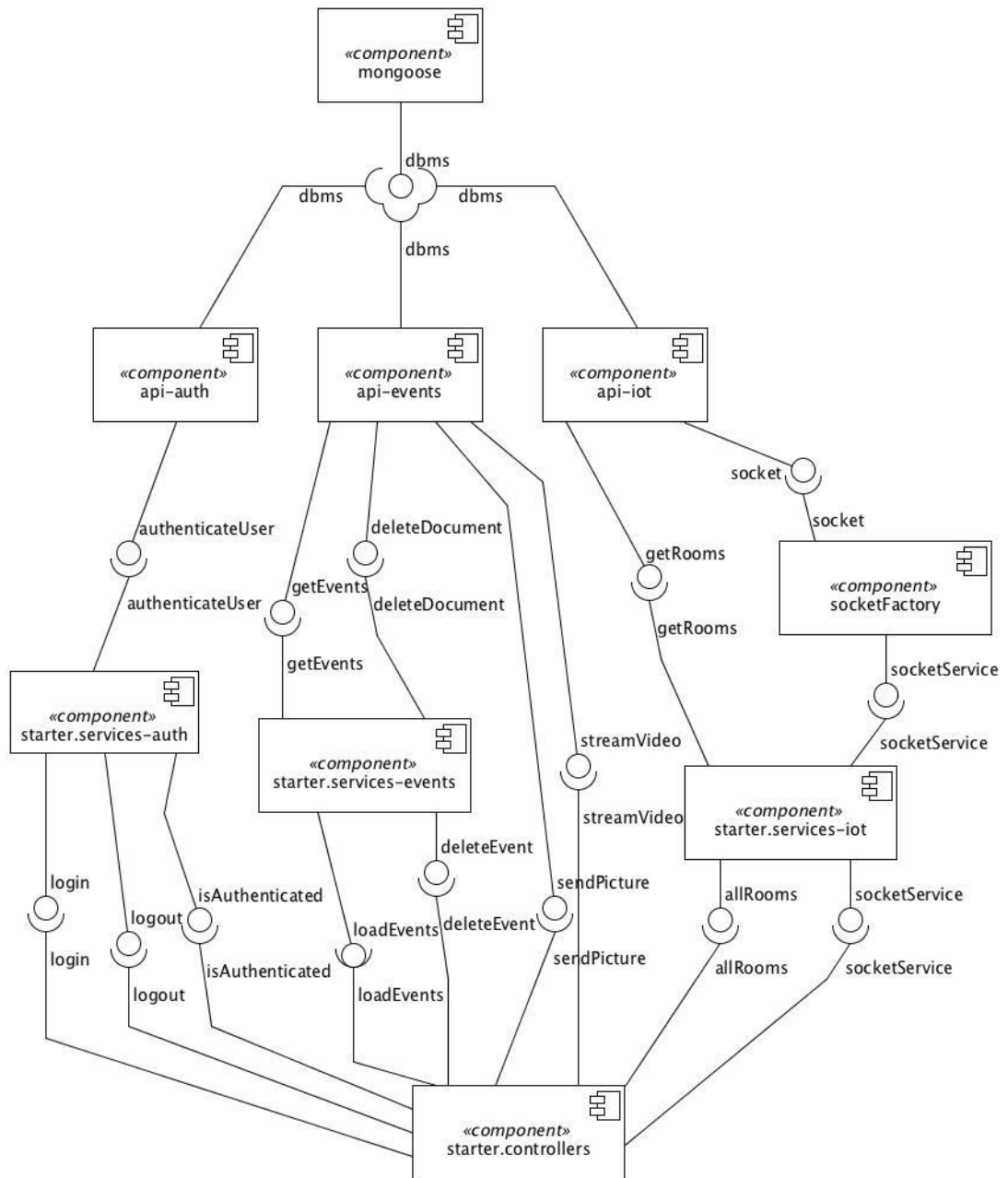


Figure 3. Components Diagram

Based on figure 3, the components prefix with “starter” are based in the front-end side of the product. The product is developed based on the MVC pattern which helps to have an easier maintenance as:

- Components prefixed with “starter.services” are the model of the MVC pattern.
- Component prefixed with “starter.controller” contains the controllers of the MVC pattern.
- The views are not shown in the components diagram.

Above the front-end side components, there is all components provided by the back-end side providing API to get the data required by the front-end from the database management system (DBMS).

4.2.3 Class Diagram

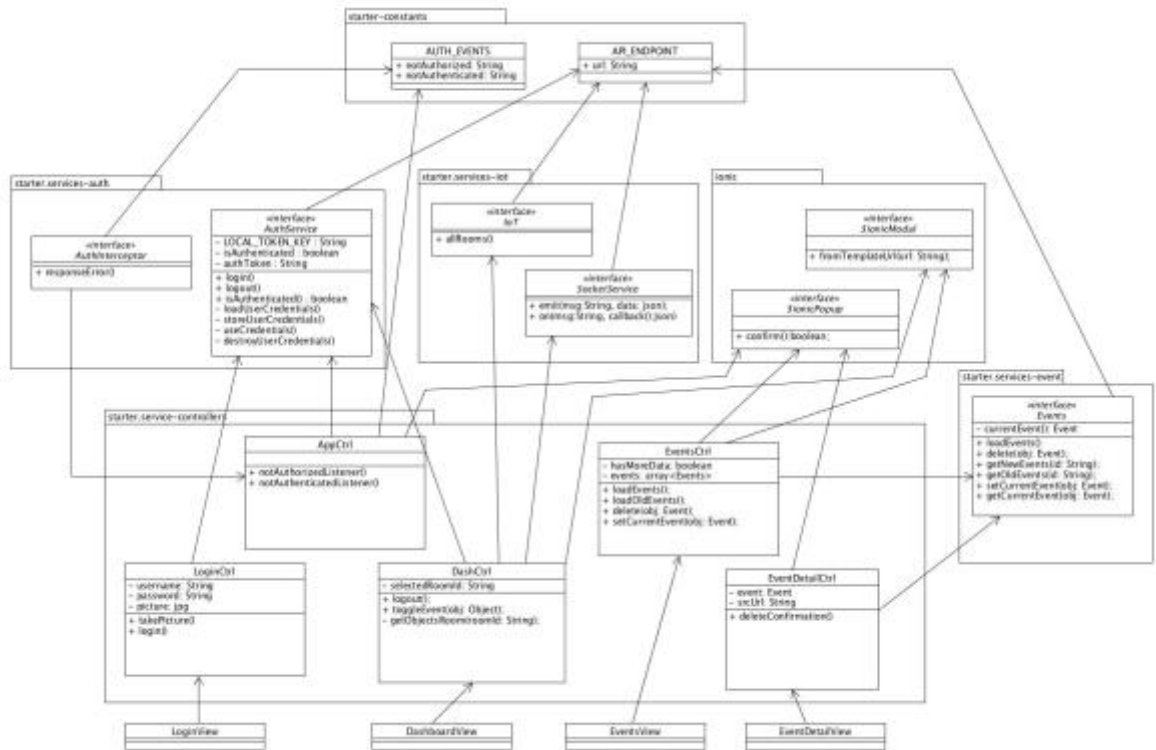


Figure 4. Class Diagram

The figure 4 hereinabove represents the class diagram of the mobile application to develop. The logical part of the product is divided into Angular.js modules. Then we have:

- **starter-constants**: contains all constants required by the mobile application such as the server IP address, sockets server IP, API IP address.
- **starter.service-auth**: this module is managing all data related to the authentication such as storing the JSON Web Token and adding it to the Authorization header.
- **starter.service-iot**: provides the real-time communication socket to manage “things” available in the indoor environment.
- **starter.service-event**: it provides the CRUD for managing the events gathered by the security camera and stored into the database.
- **starter.service-controllers**: all controllers of the application are stored inside this module and each controller of it is linked to a user interface represented as *prefixView*.
- **ionic**: this module is available by default with the Ionic framework and provides many more dependencies.

Developing the application based in the MVC architecture will make easier the task of adding features as we will not have to update any other modules except the updated one and helps to fix issues faster. We will develop the application trying to have a strong cohesion between classes and a weak coupling between modules making the MVC model even better for maintenance through the product life-time.

4.2.4 Sequence Diagram – User Authentication

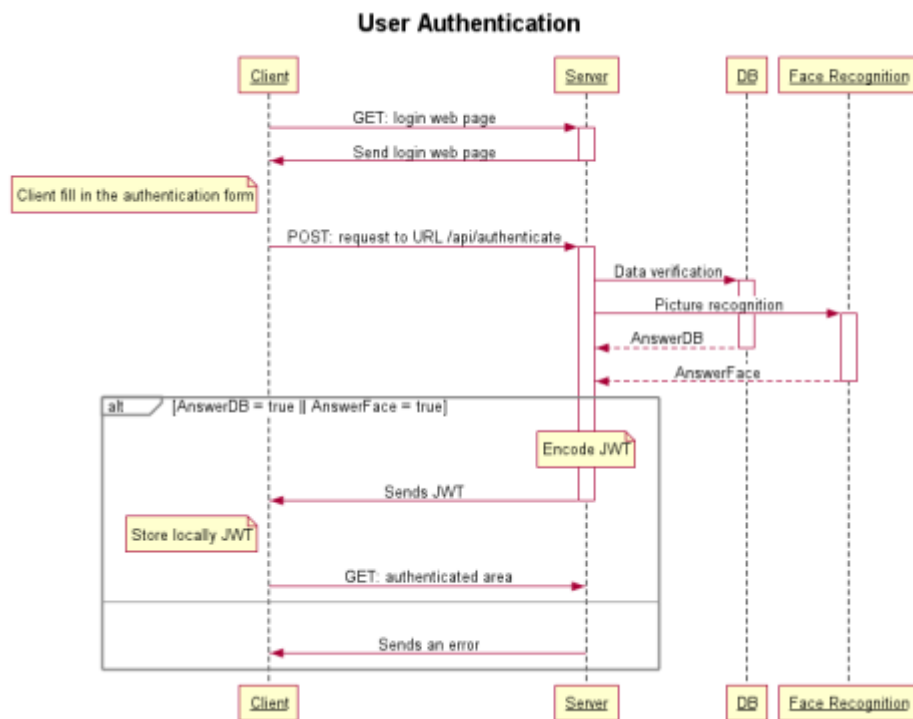


Figure 5. User Authentication sequence diagram

The user authentication is composed of four actors (see figure 5 hereinabove). The primary actor is the Client which is expecting a response from the secondary actors (Server, DB, Face Recognition).

Client, if it is a browser, will get at first the login webpage to enter a user credentials otherwise it starts directly at the post request to the authentication API URL.

Server will contain the authentication API and will ask to DB (FaceRecognition is not implemented) if Client credentials are correct. Once DB answered, Server will send back a response containing a Boolean as authentication status.

An alternative is used on Client to check Boolean value. If the authentication succeeds, the Client will store in its local storage the JSON web token (JWT) received in the response and will have access to the private area of the application. Otherwise, he will be redirected to the login page and will have to repeat the process from the post request authenticate.

5 Implementation

5.1 Pre-requirements

The software has to run on a Raspberry Pi. Then, it has to be configured and be able to run all the technologies used.

In order to be able to run the software, the Raspberry Pi requirements are:

- Node.JS & NPM
- NoSQL MongoDB
- Python 2.7
- Motion software
- FFMPEG enabled with x264³ codec

5.2 FFMPEG with x264

FFMPEG is a powerful open source library providing audio and video encoding used by multiple software's including Motion software. FFMPEG in its basic form is not implemented with the codec x264, required to encode a video into a readable format in order to be able to stream them to a video tag on HTML5. Thus, the codec has to be previously installed and enabled while building and installing FFMPEG.

We decided to use the x264 encoding format as it is the most compatible video format with current browsers (refer to table 2 hereinafter), allowing us to stream all the recorded videos to any mobile device and computer.

Browser	MP4 ⁴	WebM	Ogg
Internet Explorer	YES	NO	NO
Chrome (all devices)	YES	YES	YES
Firefox (all devices)	YES	YES	YES
Safari (iOS & OSX)	YES	NO	NO
Opera	YES	YES	YES
Android browser	YES	YES	YES

Table 2. Browsers video compatibility (<http://caniuse.com/#search=h264>)

³ x264 is an open source library for encoding video streams.

⁴ MPEG-4 (.mp4) is a video format encoded with x264 library.

5.3 Motion for Raspberry Pi

Motion is an open source software developed for Linux OS used for video detection. It uses FFMPEG open source software to encode frames from motions into a video.

Unfortunately, Motion has some software limitations:

- Motion is only able to store directly into MySQL, PostgreSQL, SQLite3 databases.
- Video flow cannot be sent to a HTTP server via a GET request in order to pipe the video flow and store it.

Fortunately, Motion has a configuration file which provides two triggers for developing a solution to the lack of features:

- on_movie_end: executes a command when a movie is closed.
- on_picture_end: executes a command when an image is saved.

Thus, to be able to use Motion as video detection software and store the events into our application database, we decided to create a shell script which will help to meet the requirements:

```
1. #!/bin/bash
   #SCRIPT created by Frederic Lopes
   #Variables
2. database="thesis-db"
3. log="script.log"
4. path="/home/pi/Desktop/shell/files/output.mp4"
5. file=$1
   #Alt to check if it is a lastsnap (we do not store lastsnaps)
6. if [[ $file == *'lastsnap'* ]]; then
7.     exit 0
8. fi
   #Alt to check if file exists
9. if [ -f "$file" ]; then
   #Alt to check if it is a video
10.    if [ "${file##*}" == 'avi' ]; then
11.        /home/pi/bin/ffmpeg -i "$file" -c:v libx264 "$path" &
12.        wait
13.        rm $file
14.        file=$path
15.    fi
16.    date=$(date +%Y-%m-%d:%H:%M:%S)
17.    type="${file##*}"
18.    filename=$date.$type
19.    mongofiles -d $database --type $extension --local $file put $filename
20.    rm $file
    exit 0
21. else
22.    echo "ERROR: NOT FOUND $file" >> $log
23.    echo "ERROR: LIST OF FILES IN ./files: >> $log
24.    echo $(cd ./files | ls -la)" >> $log
25.    exit 1
26. fi
```

Figure 6. Bash Script developed for filling in missing requirements from Motion

Based on figure 6, from row two till the row five, we are setting the attributes required such as the database name, the log file to write the errors of the script, the path to the folder

where the files are saved and the parameter received when the script is launched. The parameter is the name of the file saved into files folder and sent by Motion while calling the bash script.

On row six, we are testing if the file is a lastsnap. A lastsnap is a picture saved in order to process pixels' comparison with new frames. Thus, lastsnap do not show any motion and are not relevant for storing them into the database and pushed to users' device.

On row nine, we test if the file received as parameter exists otherwise we output an error into the script log file for further analyzes.

If our file exists, we check if it is a video. This point is crucial as Motion is only able to encode into MPEG-4 which is not supported by HTML5 video tag. Thus, we launch an encoding process receiving as input the MPEG-4 file and outputting a x264 encoded video required for HTML5 video tag.

Then, from row six-teen till twenty, we store the document into our database using GridFS, which creates chunks of the recorded document in order to store our records as MongoDB has a document size limitation of 16mb.

5.4 Security

Developing a security system for an indoor environment means that private data will be sent through the internet. Then, it is mandatory to ensure that users are allowed to access to private data and that the data is reliable and not modified by someone. This aspect of the security allows us to understand that using the simple HTTP is not a viable solution for the product as it sends all network frames in clear-text making them readable by anybody.

Security is a complex domain and going through all security leaks is not viable for this thesis, as security is an ongoing process, but some protections can be implemented in order to avoid basic security leaks:

- Hashing users' password in the database ensuring that if data is stolen from the server, the pirate is still not be able to log in.
- Headers reinforcement will help us to protect against XSS⁵ as we are using sockets for the real-time communication.
- **HTTPS is not an option** as we do not want that data to be readable over internet.

⁵ Cross-site Scripting (XSS) is an attack which consists in sending malicious script to an unsuspecting user over.

Apache Cordova is providing an interesting security guide and is going through security best practices that you can see on their documentation webpage.

(<https://cordova.apache.org/docs/en/4.0.0/guide/appdev/security/index.html>)

5.4.1 Hash

Nowadays, protecting data is a major challenge and ensuring that hackers do not access to private information is an everyday challenge as there is no system protecting all attacks.

Thus, to protect some data stored in the database, we can use encryption algorithms in order to make it unreadable for any user who could access to it. One of them is the cryptographic hash function which is a powerful algorithm to encrypt data with a one-way function which means that the data cannot be un-hashed once done.

It is a best practice to store user password in an encrypted way as storing such data in its original form can conduct to a lack of security if an attacker steals the data from the database. Thus, encrypting the password will ensure that even if the database is stolen, the attacker will not be able to use any password to login into the system.

To implement this encryption, we simply used the dependency *bcrypt* which provides an asynchronous encryption method that we used in a middleware executed before saving a new user into the database.

```
1. var bcrypt = require('bcrypt');
2. UserSchema.pre('save', function (next) {
3.   var user = this;
4.   if (this.isModified('password') || this.isNew) {
5.     bcrypt.genSalt(10, function (err, salt) {
6.       if (err) {return next(err);}
7.       bcrypt.hash(user.password, salt, function (err, hash) {
8.         if (err) {return next(err);}
9.         user.password = hash;
10.        next();
11.      });
12.    });
13.  } else {return next();}
14. });
```

Figure 7. Implementation of hash process to encrypt user's password.

Based on figure 7, first row is the call of the dependency *bcrypt* which will be used to encrypt data. On second row, we are using an object created in our user model which provides a trigger to do pre-modifications on a model based on an event named as the first parameter of the method (e.g. in our figure, *save* is the trigger event name). On row 7, we are then hashing the user's password and calling then the method *next()* which is a simple callback sending back to the main process which triggered *pre('save', ...)*.

5.4.2 Helmet

Helmet.JS is a collection of eleven middleware functions that set HTTP headers. It helps to secure an Express application for Node.JS and prevent some headers security leaks.

Header	Included by default?
Content-Security-Policy	NO
X-DNS-Prefetch-Control	YES
X-Frame-Options	YES
X-Powered-By	YES
Public-Key-Pins	NO
Strict-Transport-Security	YES
X-Download-Options	YES
Cache	NO
X-Content-Type-Options	YES
Referrer-Policy	NO
X-XSS-Protection	YES

Table 3. Helmet headers reinforcement features (helmetjs.github.io/docs/)

As we can see on table 3, using Helmet is covering many security features for HTTP headers but will not protect us against all kind of attacks and may require extra security implementations such as SQL injection.

```
1. var express = require('express');
2. var helmet = require('helmet');
3. var app = express();
4. app.use(helmet({noCache: true}));
```

Figure 8. Implementation of Helmet dependency to protect header.

On figure 8, we are simply using Helmet dependency to protect HTTP headers of an Express.js application. On row 4, we are adding Helmet middleware to our express application and enabling the Cache module which concerns headers:

- Cache-Control: used to tell the browser to cache responses for a specific time.
- Surrogate-Control: eschew caching intermediate caches.
- Pragma: similar to Cache-Control but has a better support for old browsers.
- Expires: specifies to the browser when the cache content is considered as out of date.

5.4.3 HTTPS

We discussed about how important it is to protect the data stored in the database and how we implemented it but protecting data over internet is even more important. Using HTTP is not a viable solution to communicate, even more if private data such as passwords and

usernames are sent. Then, it is necessary to protect all the data flow from being read by a MITM⁶.

HTTPS has been developed combining HTTP and SSL/TLS encryption process to ensure that:

- The data flow is encrypted and cannot be read by a third party.
- The identity of the second party is known and sure to be trusted.

This protocol is working with two keys system to provide a symmetric and asymmetric encryption solution.

A symmetric encryption uses the same key to encrypt and decrypt the data (refer to figure 9 hereinafter). It means that the key cannot be send over internet in a clear way otherwise any MITM can catch it and read all the data flow.

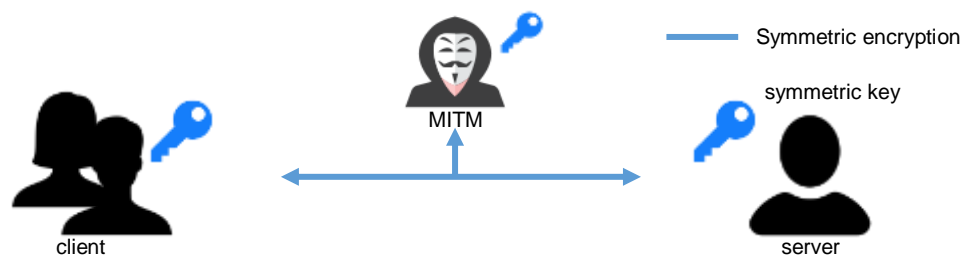


Figure 9. symmetric encryption dataflow

An asymmetric encryption, compared to the symmetric, is working with two keys:

- A public key which encrypts the data and cannot decrypt it.
- A private key which is able to decrypt the data encrypted by its public key.

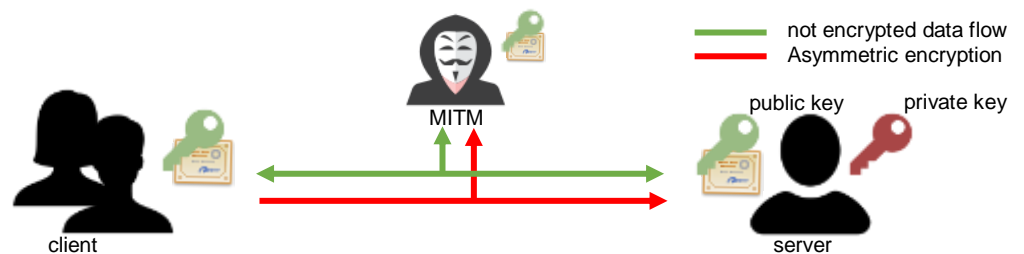


Figure 10. asymmetric encryption dataflow

Based on figure 10, using an asymmetric key is providing a one-way encryption (represented by the red arrow) done with the public key received previously (represented by the green key and the green arrow). Furthermore, encrypting the data does not ensure the identity of the server. Thus, we need to combine our public key with trustworthy

⁶ Man-in-the-middle (MITM) is an attack where the attacker relays and possibly alters the communication between two computers.

certificate signed by a Certificate Authority (CA) ensuring our identity to the client otherwise the communication is vulnerable as a MITM can easily create a self-signed certificate. All modern browsers display HTTPS using a certificate that isn't trustworthy (e.g. self-signed or overdue) as an unsecure connection, vulnerable to MITM attacks (see figure 11 hereinafter). (Howard, n.d.)

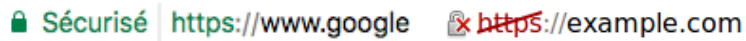


Figure 11. Authority-signed certificate versus Self-signed certificate

HTTPS is then a reliable protocol to encrypt all the data flow and ensure the identity of the server side as it is combining the asymmetric encryption process, which is a relatively slow algorithm, with the symmetric encryption. HTTPS is processed by following the steps below, named Handshake protocol, and based on figure 12 hereinafter:

1. Client asks for a SSL connection with the server.
2. Server response with the SSL certificate which includes the public key.
3. Client approves the certificate and public key.
4. Client generates a symmetric key.
5. Client encrypts the generated symmetric key with the asymmetric public key received in the certificate and send it to the server.
6. Server decrypt the encrypted symmetric key with its private key.
7. SSL session is established.

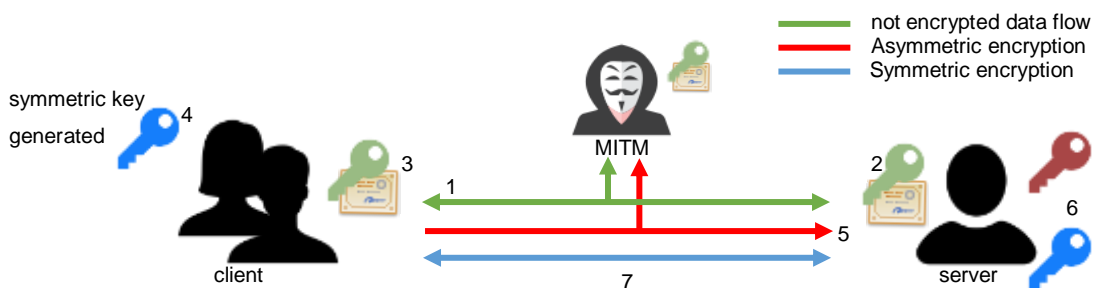


Figure 12. HTTPS communication flows

Node.JS over HTTPS

For our product, we are going to use a self-signed certificate as we do not have a domain name for our server. It is quite simple to get a signed certificate as it can be released by the CA "Let's Encrypt" which is a CA open source and then, free of charge.

Node.JS has HTTPS as core module and does not require any external dependency to be used. It receives an object composed of two attributes which are the key and a certificate string that we have generated using OpenSSL command line tool.

```

1. var https = require('https');
2. var fs = require('fs');
3. var app = require('./app');
4. var https_options = {
5.   key: fs.readFileSync('./ssl-certificates/key.pem'),
6.   cert: fs.readFileSync('./ssl-certificates/certificate.pem')
7. };
8. var server = https.createServer(https_options, app).listen(port);

```

Figure 13. HTTPS implementation over Node.js.

Based on figure 13, we are simply listening, on row 8, to a configured Express application, imported on row 3, using the core module https imported on row 1. In order to use https module, we need to add as first parameter, set in a variable on row 4 to 7, of the server the *https_options* where we will indicate the path to the SSL certificates.

5.5 Internet of Things

Controlling indoor objects requires external components such as copper wires, resistors, relays, etc... For our implementation of the connected objects, we are working with low voltage components combined with Python language in order to control the GPIOs of the Raspberry Pi on run time.

Before going through the implementation of IoT solution for our project, we advise that the implementation has been calculated previously to avoid any risks related to high amps on the electrical circuit.

5.5.1 Required components

As introduced, connecting objects requires external components. As we are working with electronic components, we have to take care of voltages used and amps generated, otherwise we can destroy all our circuit, Raspberry Pi included.

To implement an example of IoT on our product and reduce risks of destruction, we only used low voltage components. These components can be easily purchased on the internet at a cheap price.

The components required for the implementation are:

- A solderless breadboard as testing construction base simulating a room.
- Wires for connecting electrically all components to the Raspberry Pi.
- Resistors to protect LEDs against high amps.
- LEDs requiring around 3 volts and handling approximately 20miliAmps (mA). These features are different regarding the colour of each LED used.

The GPIOs of the Raspberry Pi are able to provide 3.3v, a maximum recommended of 16mA per pin and a total of 50mA for all pins. Thus, we only are implementing two LEDs to make an electrical schema as simple and safe as possible for the Raspberry Pi.

5.5.2 Electronic diagrams

In order to implement our LEDs and switch them ON and OFF with Python, we design a simple electrical circuit with two LEDs and resistors to reduce the amps of the circuit.

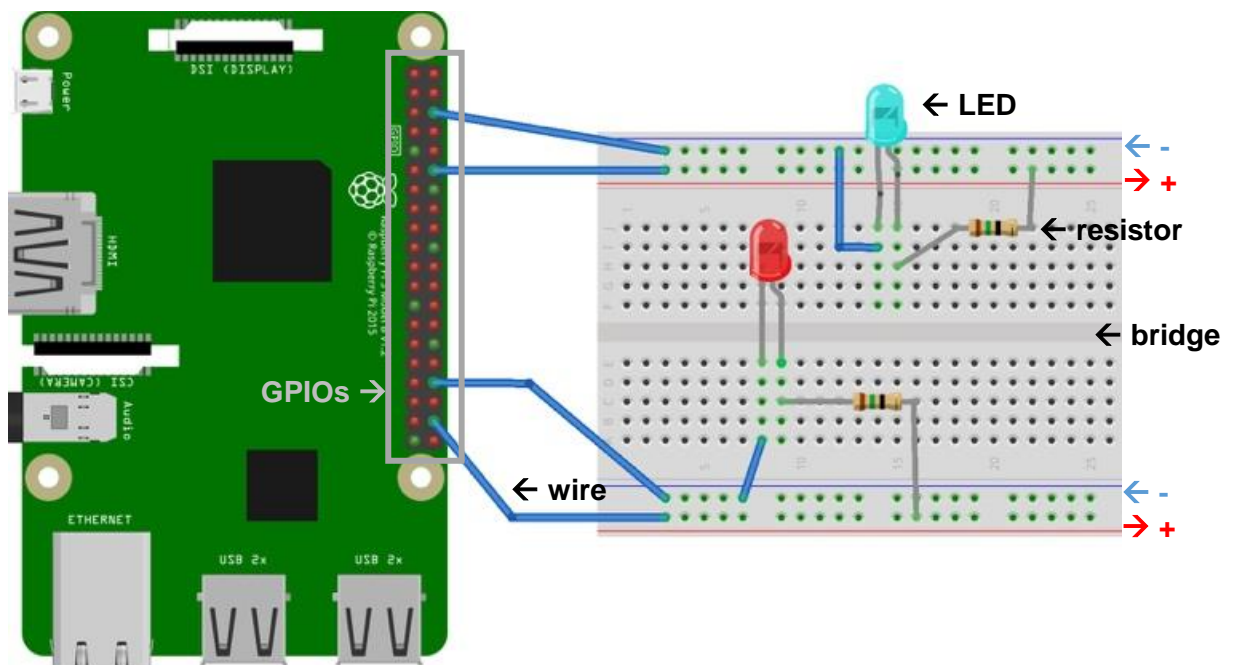


Figure 14. Wiring design.

Based on figure 14, we implemented two LEDs (blue and red) powered by the Raspberry Pi GPIOs to switch them ON/OFF using a Python script.

The red line represents the conventional current (positive +) and blue line the ground current (negative -) or electrons current. As you can see on the solderless breadboard, the green points are those who have a current.

We connected two GPIOs to a solderless breadboard, using female to male wires, on two different testing sides separated by a bridge. The bridge does not let the current cross to the other side and allows then to simulate two different rooms.

For each LED, we added a resistor to reduce the intensity of the circuit and avoid destruction of our LED. As indicated in chapter 5.5.1, a LED can support a 20mA and requires a voltage of ~3v, thus, we need to use resistors to control the intensity of the circuit.

Finally, each circuit is ending on a ground pin of the Raspberry Pi to close it and let the electrons light our LEDs.

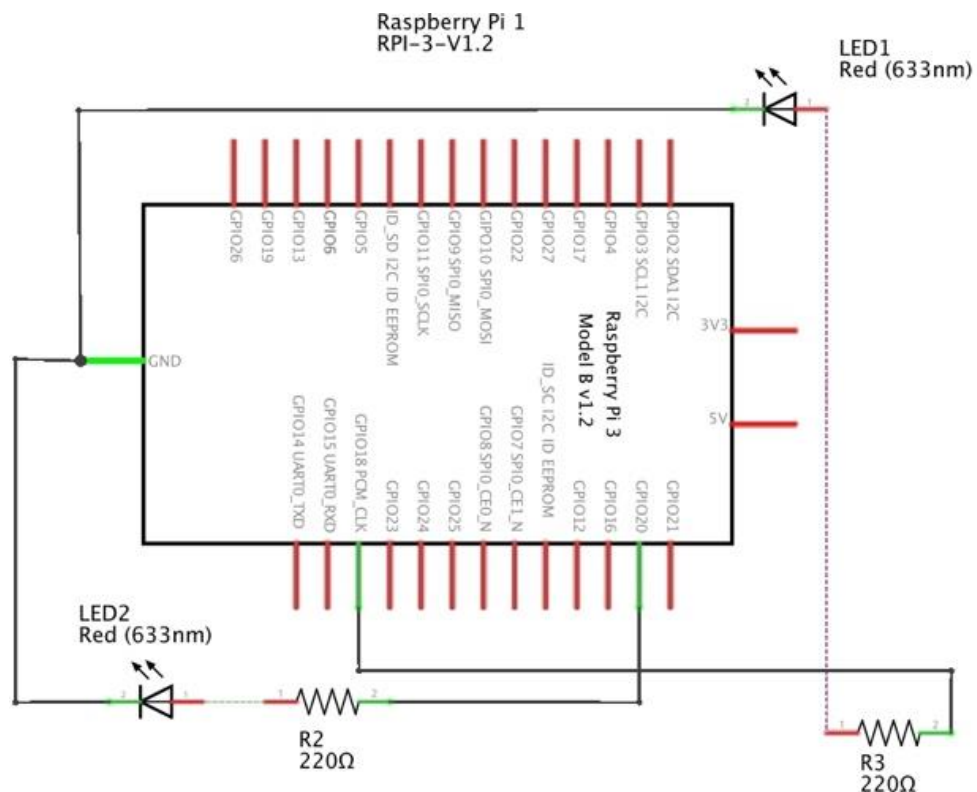


Figure 15. Electrical diagram of Raspberry Pi GPIOs.

As depicted by figure 15, we have an electrical diagram of our Raspberry Pi GPIOs and the features of the electrical components used. Based on this diagram, we calculated the amperes of the circuit and ensured that it was safe for the electronic components and our Raspberry Pi.

5.5.3 GPIOs controlled over Python

The Raspberry Pi GPIOs can be switched ON/OFF and have a Direct Current (DC) of 3.3 volts. Based on the electrical design of figure 15, we can notice that each GPIO has a number and for each wiring you may know to which GPIO you are connected, otherwise you can use a default 5v pin and destroy your circuit.

```

1. #!/usr/local/bin/python
2. # -*- coding: utf-8 -*-
3. import sys, json

4. def read_in():
5.     lines = sys.stdin.readlines()
6.     #As our input has only one JSON line, return parse JSON data
7.     return json.loads(lines[0])

```

Figure 16. Tools.py

On figure 16, we are listing all reusable methods that can be used by other python scripts. We are then applying the Don't Repeat Yourself (DRY) principle. The method on row 4 is reading the input received while calling the script and parsing the JSON data read.

```
1. #!/usr/local/bin/python
2. # -*- coding: utf-8 -*-
3. from tools import *
4. import RPi.GPIO as GPIO

5. def _launch(state, gpio):
6.     GPIO.setmode(GPIO.BCM)
7.     GPIO.setwarnings(False)
8.     GPIO.setup(gpio, GPIO.OUT)
9.     if (state):
10.         GPIO.output(gpio, GPIO.HIGH)
11.     else:
12.         GPIO.output(gpio, GPIO.LOW)

13. def _main():
14.     #get our data as an array from read_in()
15.     lines = read_in()
16.     #objects array has to have a length of 2
17.     state = lines[0]
18.     if len(lines) == 2 and state or not state:
19.         if lines[1] is not None:
20.             gpio = strToInt(lines[1])
21.             _launch(state, gpio)
22.         else:
23.             print("INPUT INCORRECT!")

24. if __name__ == '__main__':
25.     _main()
```

Figure 17. gpioUpdate.py

On figure 17, we are controlling two states objects and updating their current state. On row 3 and 4, we are importing the libraries required to run the script (notice: on row 4, *RPi.GPIO* library is only available on a Raspberry Pi).

From row 13 to 21, we are starting our process by getting at first, on row 14, the arguments while python script was called. Then, as this python script is only to turn ON/OFF a GPIO, we are ensuring that the parameters received are correct before changing GPIO state on row 19.

From row 5 to 12, we change the state of a GPIO with the parameters received while calling the method. Then, a Boolean will switch between a GPIO output of *GPIO.HIGH* to *GPIO.LOW*. Those two constants are provided by *RPi.GPIO* library to control Raspberry Pi pins.

5.6 Ionic Application User Interface

The UI of the application is composed of four main parts

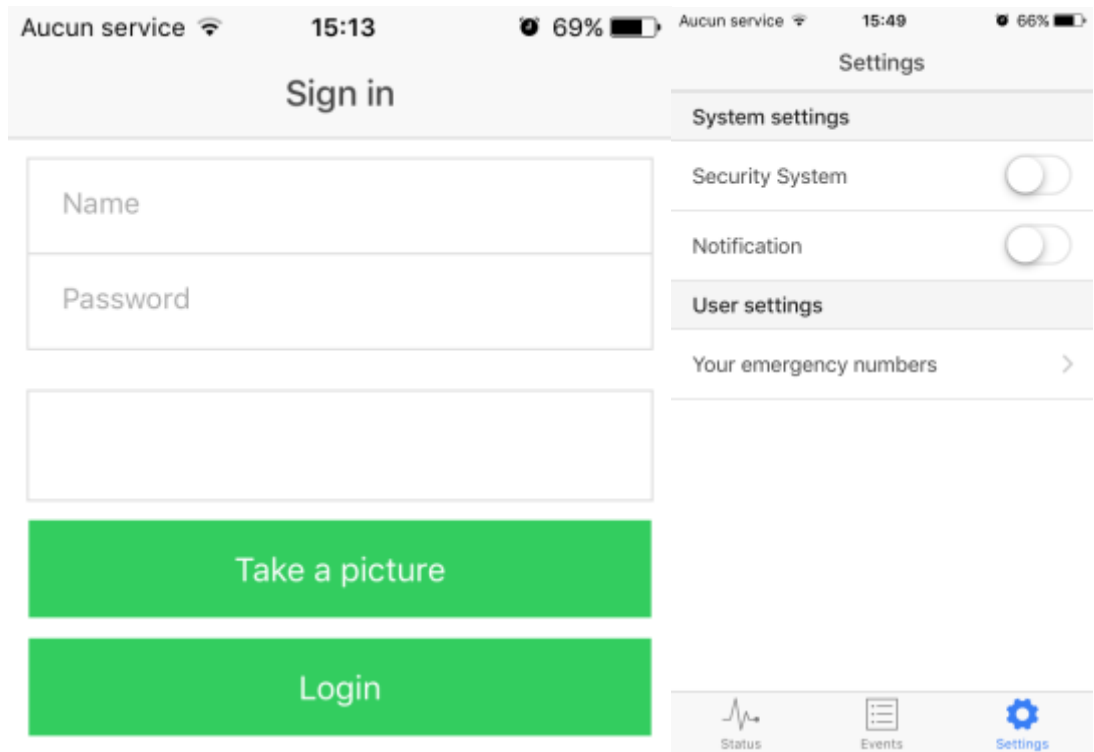


Figure 18. Sign in view & settings tab

On figure 18, the user has to sign in to access to the authenticated area of the application. The button "Take a picture" activates by default the front camera of the device in order to take a picture of the user for a future authentication process with face recognition.

The second figure is the settings tab, only accessible to an authenticated user. The user will have the possibility to enable the security system (turn ON/OFF the security camera).

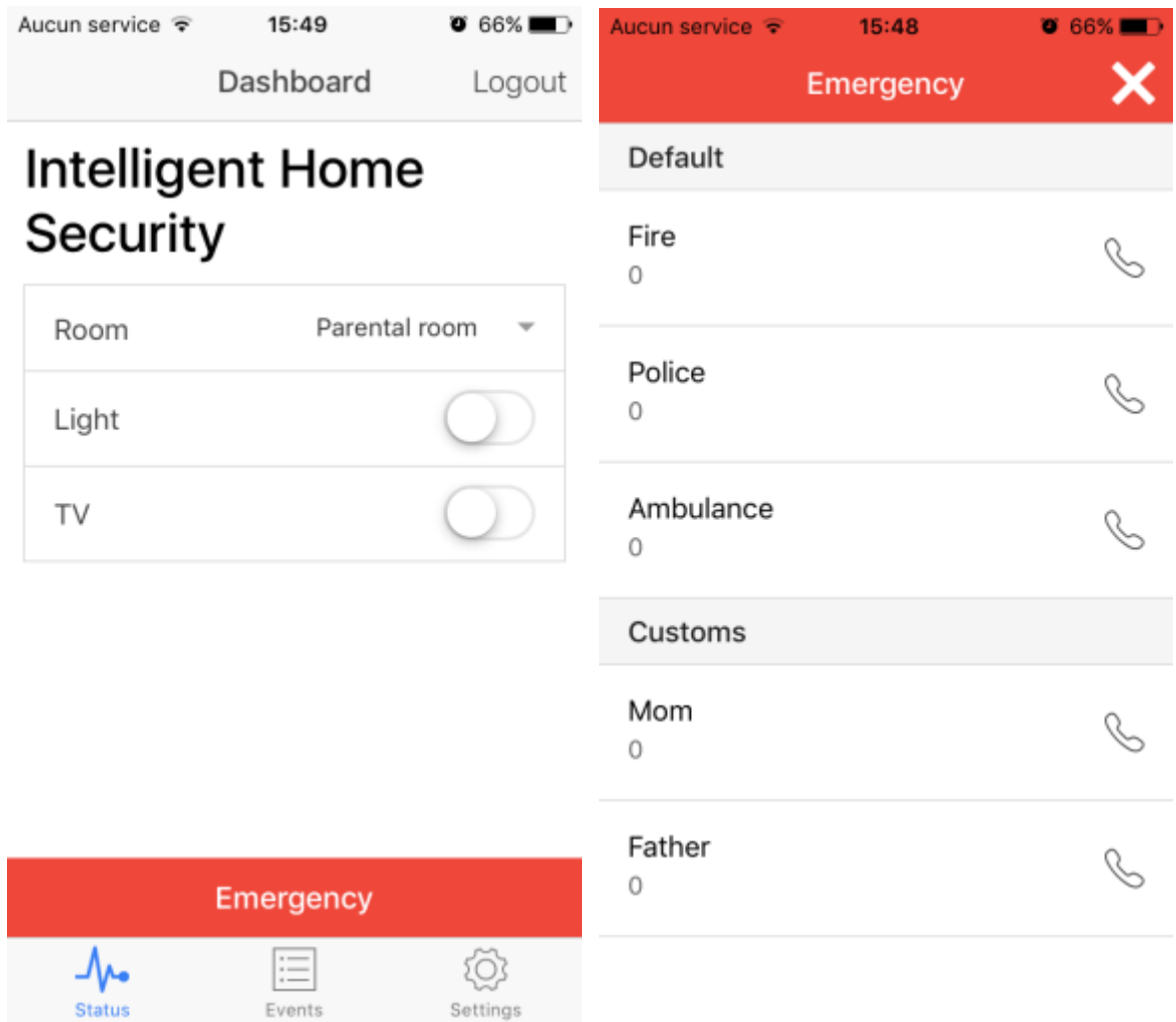


Figure 19. Status tab with IoT control panel & modal emergency numbers

On figure 19, the user is in an authenticated area and has access to the IoT control panel from which he can interact with connected objects and know their current state.

The emergency button opens a modal window, as represented in the right picture, with a quick access to two types of emergency numbers:

- Default numbers are those known by the server regarding to the country localisation of the indoor security system. Those are downloaded once authenticated and stored in local storage.
- Customs are those added by the user himself and can be customizable.

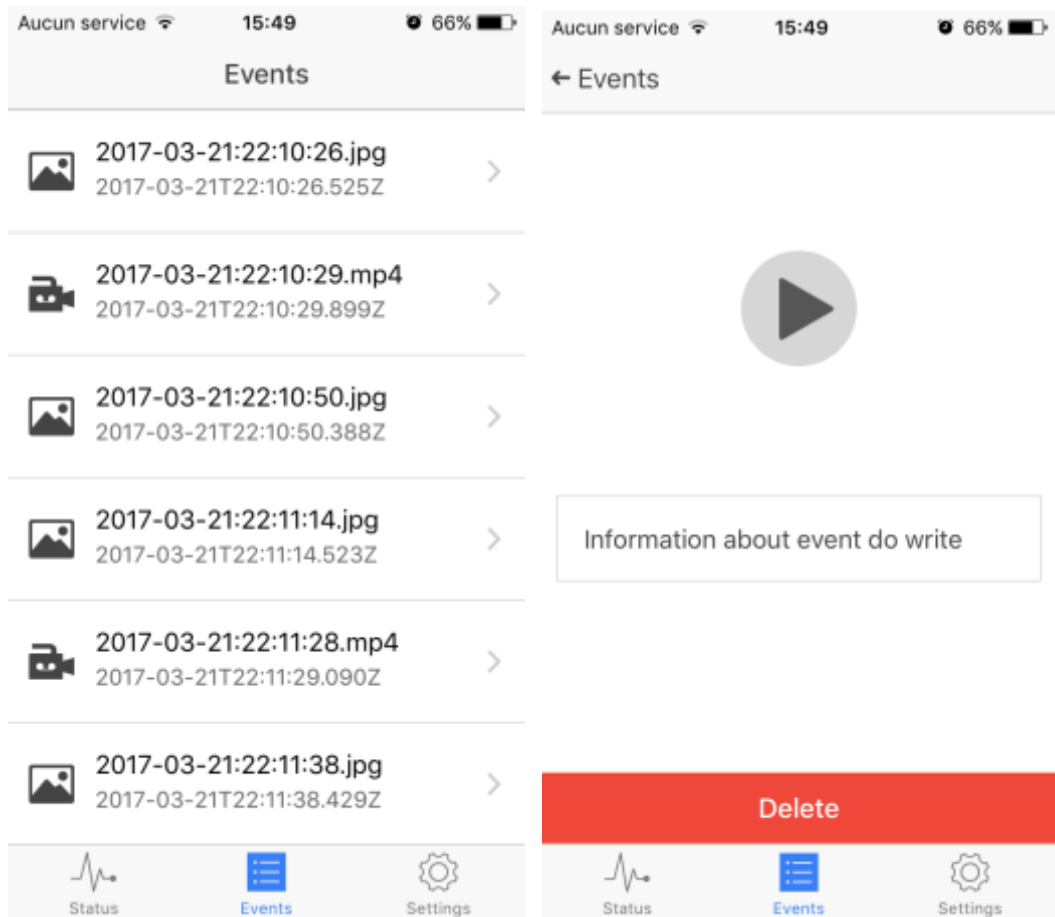


Figure 20. Events tab & event details view

The figure 20 displays the events gathered by the security system. It is from this view that the user will do representational state transfer (REST) commands such as delete an event from the database, or read it through a pipe as represented on the event details view.

6 Testing

As our developed product is based on APIs, we are going to test them using an application called Postman which is used to test POST/GET/DELETE/UPDATE requests, headers parameters and many other features.

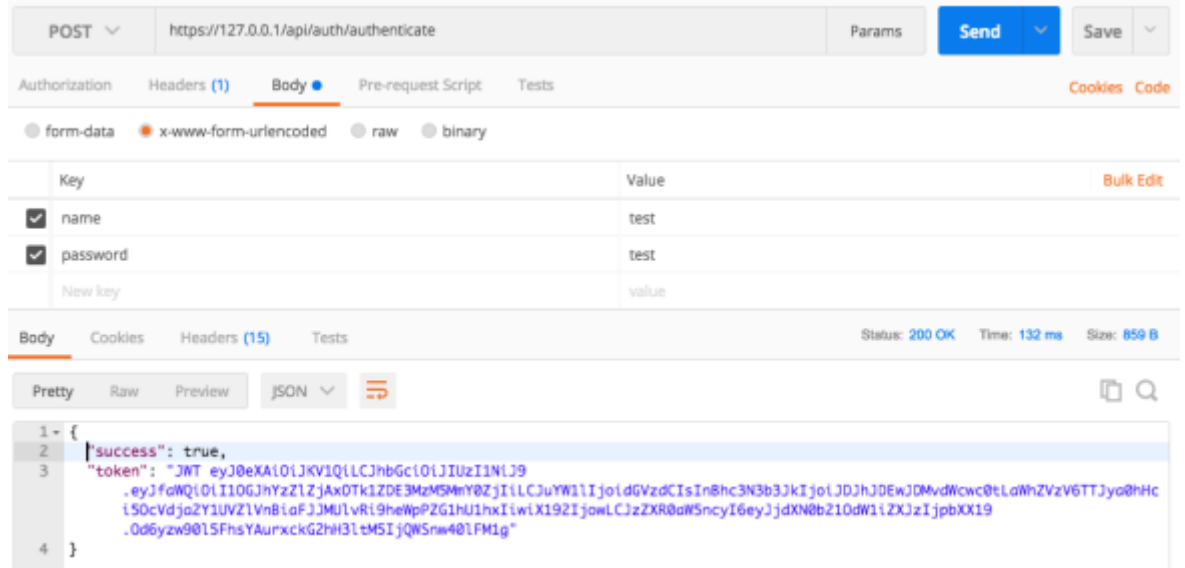
We are also going to use Wireshark network protocol analyzer which is an open source software able to listen to all network protocols and see the data flow.

Thus, we should cover many technologies implemented in the product and ensure, by sending requests, listening to the communication, and reading responses, that the software is behaving as expected.

6.1 JSON Web Token

As we are using JWT as session, we are going to test that we receive a proper JWT encoded with a signature that has to be match with the secret used while generating the token.

To realize this test, we used Postman and accessed to the authentication API which is in charge of verifying a user credentials and responding with a JSON object. Our database has only one user registered with the credentials *test* as username and password.



The screenshot displays a Postman interface for a POST request to the endpoint `https://127.0.0.1/api/auth/authenticate`. The request body is set to `x-www-form-urlencoded` and contains the following data:

Key	Value
name	test
password	test

The response is a JSON object with the following structure:

```
1 - {
2   "success": true,
3   "token": "JWT_eyJ0eXA101JV1Q(LCJhbGct01JlUzI1NlJ9
4   .eyJfaWQiOiI1OGJhYzZlZjAxOTk1ZDE3MzZmM5MmY0ZjIiLCJyYW11IjoIdGVzdCIsIn8hc3N3b3JkIjo1JDJhJDEwJDMvdWcwc0tLaWhZVzV6TTJya0hHc
5   L50cVdjoZy1UVz1VnBiaFJJMUlvr19heWpPZG1hU1hkIiw1X192IjowLjZzZXR0aW5ncyI6eyJjdXN0b210dW1iZXJzIjpbXX19
6   .0d6yzw9015FhsYAurxcckG2Hh31eM5IjQWSnw401FM1g"
7 }
```

Figure 18. POST request to authentication API & server response.

Based on figure 18, we are doing a POST request to our API `authenticate` and providing a body with the credentials of our registered user. As we can see on the bottom of the figure, we received a successful JSON response with two attributes. The attribute *success* is a simple Boolean giving the status of the response given by the server (authentication failed or passed). A second attribute is containing our JWT that will be stored in the local storage of the device and added to the Authorization header.

As we should only be able to access to other APIs with a valid token, we tested if the server is responding successfully to other requests or not. Then, we tried to access to the IoT API and get the list of the connected objects.

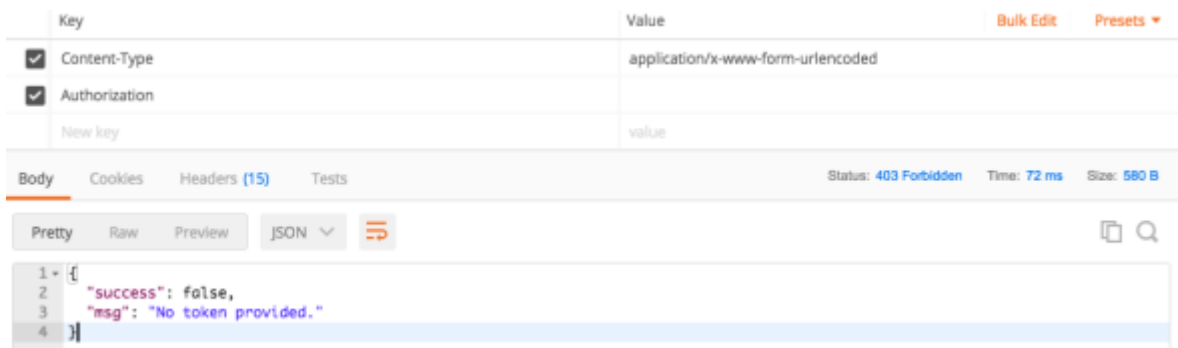


Figure 19. IoT API access without token in the Authorization header.

As depicted on figure 19, we can see that the server respond with a failure. Thus, we cannot access to any other API without providing a valid token.

We repeat the same request but by providing the previous token received as response on figure 18.

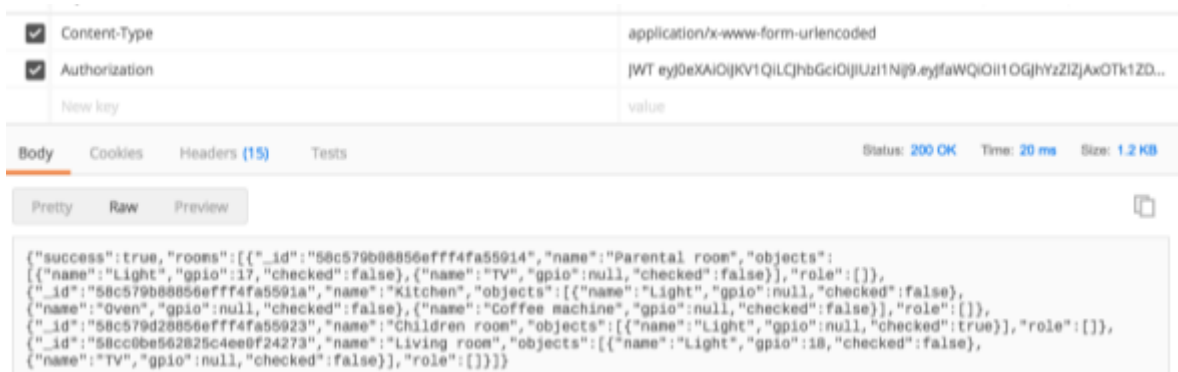


Figure 20. IoT API with valid token.

Based on figure 20, we add our token to the Authorization header which result on a success JSON response containing all connected objects of the indoor environment.

This process is the same for all APIs, except the authentication, and is working with a middleware on server side which is verifying for each client requests if there is a token and its validity before executing any event or sending data as response.

6.2 Internet of Things

The testing phase of IoT implementation has been done by testing a Passive InfraRed (PIR) sensor over the Raspberry Pi as shown on figure 21 hereinafter. A PIR sensor uses three wires, two of them are part of the electrical circuit (VCC & GND) required to run the component, the last one is an OUT wire that can be read simply by testing if there is an electric signal coming out from it (motion detected) or no signal (not motion).

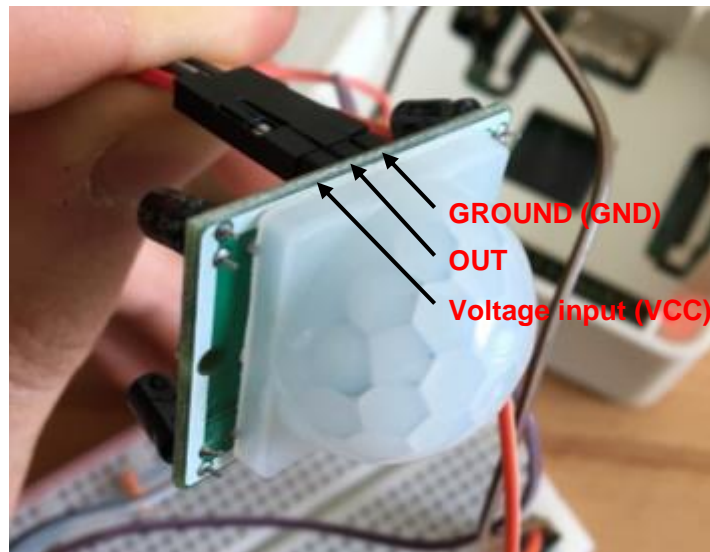


Figure 21. PIR sensor

To realize this test, we created a new python script that receives two arguments, the first argument is the GPIO pin to read in the OUT signal emitted by the PIR sensor on motion detection, the second is the GPIO pin used to output the electrical signal to run the component.

```

1. from tools import *
2. import RPi.GPIO as GPIO
3. import time

4. def _launch(pins):
5.     pinIn = pins[0]
6.     GPIO.setwarnings(False)
7.     GPIO.setmode(GPIO.BCM)
8.     GPIO.setup(pinIn, GPIO.IN) # Read from OUT wire from PIR motion sensor
9.     GPIO.setup(pins[1], GPIO.OUT) # Electrical signal, VCC wire
10.    GPIO.output(pins[1], 1) # Enable electrical signal
11.    try:
12.        while True:
13.            if GPIO.input(pinIn):
14.                print "Intruder!"
15.            else:
16.                print "No Intruder detected."
17.                time.sleep(0.5)
18.    except KeyboardInterrupt:
19.        GPIO.cleanup() # clean up GPIO on CTRL+C exit
20.        GPIO.cleanup() # clean up GPIO on normal exit

21. if __name__ == '__main__':
22.     pins = read_in()
23.     if len(pins) == 2:
24.         _launch(pins)

```

Figure 22. pir_motion.py

Based on figure 22, the python script start on line 21 where we read the arguments while launching the python script. Then, from line 4 to 9, we are setting the GPIOs that are going to be used, indicating the output pin (line 9) and input pin (line 8) in charge of read in the electrical signal emitted by the PIR sensor.

On line 11, we realize a try and catch exception. As running python scripts from Node.js creates a new bash process, it does not allow us to push data through a running python process, then, we catch a SIGINT (interrupt from keyboard) signal in order to turn off the GPIOs used with the command on line 19 and 20.

Our test concluded on a working motion detection, catching motions appearing in the detection area of the PIR sensor and printing the message on line 14. This conclusion opens the product development to multiple connected objects such as humidity, moisture, smoke or heat detectors to secure and automate an entire house.

6.3 HTTPS

In order to test that HTTPS is working properly, we used Wireshark network protocol analyzer which is an open source software able to listen to all network protocols and see the data flow. This test is done to ensure that HTTPS protocol is working properly and all data flow between client and server is encrypted and not readable by a MITM.

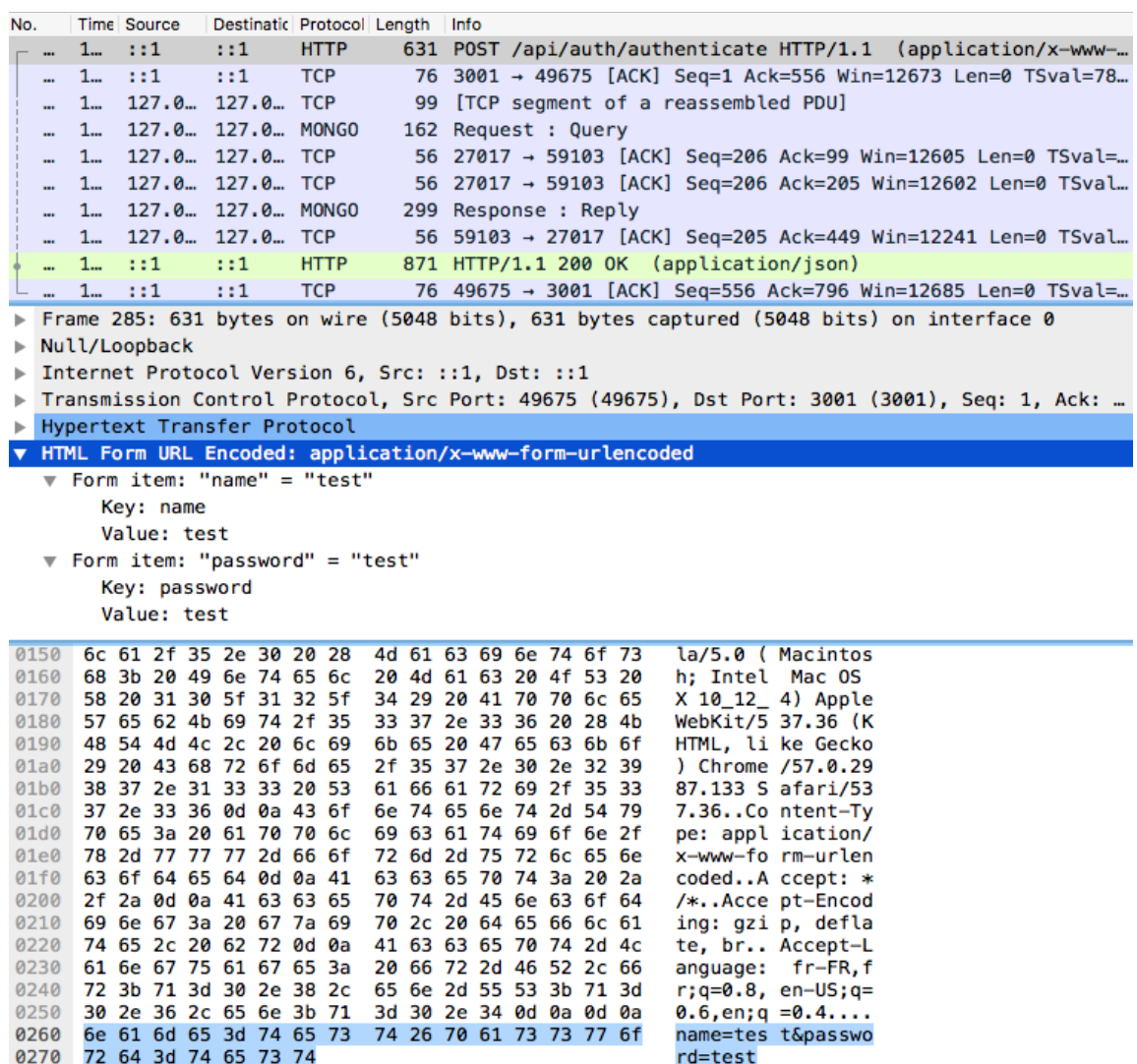


Figure 23. Wireshark output of a HTTP POST request for a user authentication.

On figure 23, we authenticated to the server using HTTP. As we can see on the bottom, highlighted in blue, the data is not protected and a MITM can stole credentials easily and see all communication between client and server and consequently, stole the JWT of a user and access to the server.

We are now going to prove that HTTPS is encrypting all the data flow between the two parties' communication.

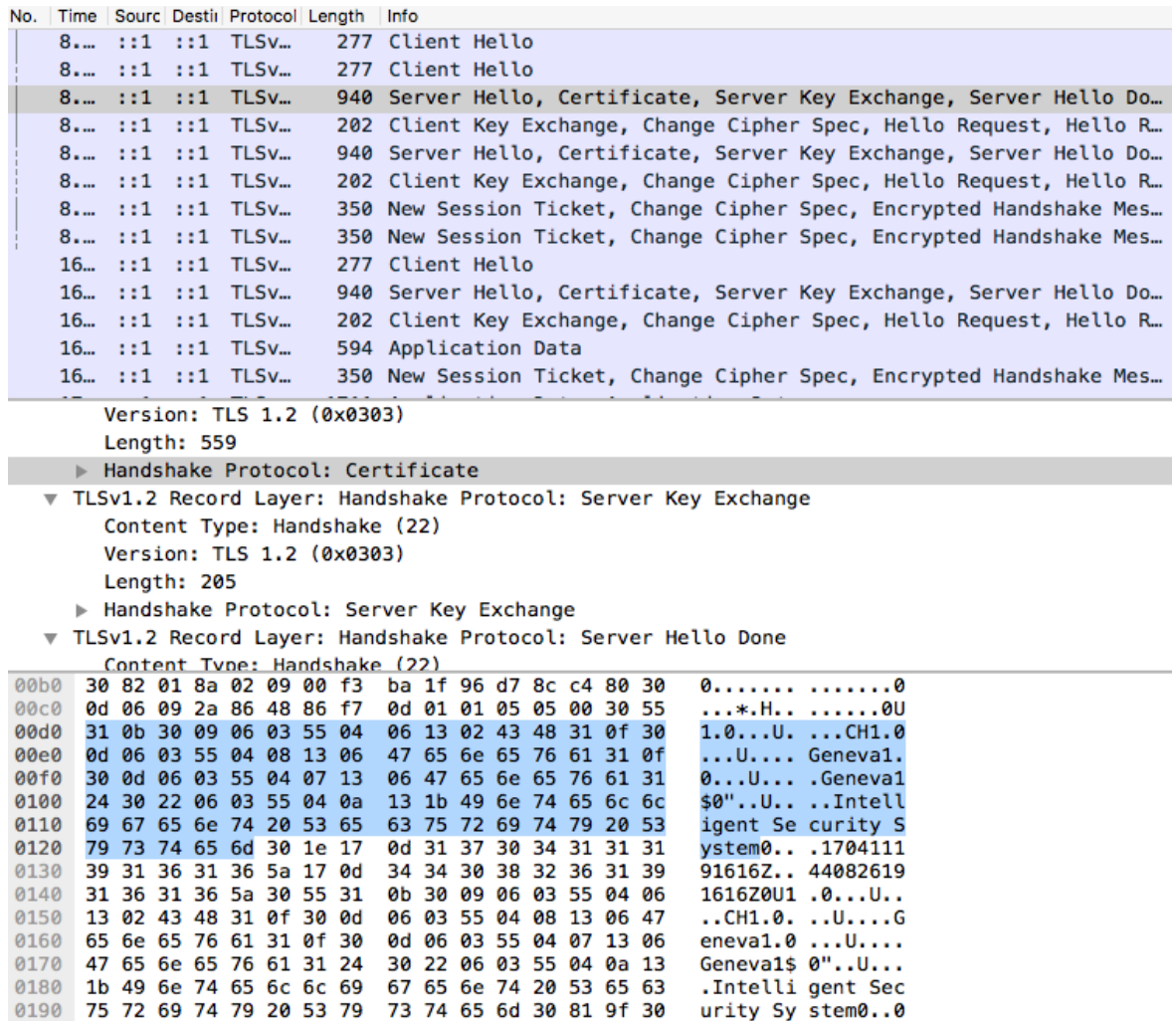


Figure 24. SSL Certificate and handshake message.

On the figure 24, highlighted in blue, we can see the server response sending its certificate with its information such as the country, state, company name, etc.. and the public key for asynchronous encryption.

The certificate cannot be encrypted as we mentioned on chapter 5.4.3 but all the data flow following the first response will be succeed with the encrypted synchronous key which will end up with a handshake message as confirmation of a proper encrypted communication between Client and Server.

No.	Time	Source	Destination	Protocol	Length	Info
...	4...	::1	::1	TLSv...	593	Client Hello
...	4...	::1	::1	TLSv...	940	Server Hello, Certificate, Server Key Exchange, Server Hello Done
...	4...	::1	::1	TLSv...	202	Client Key Exchange, Change Cipher Spec, Hello Request, Hello Request
...	4...	::1	::1	TLSv...	350	New Session Ticket, Change Cipher Spec, Encrypted Handshake Message
...	4...	::1	::1	TLSv...	593	Client Hello
...	4...	::1	::1	TLSv...	940	Server Hello, Certificate, Server Key Exchange, Server Hello Done
...	4...	::1	::1	TLSv...	202	Client Key Exchange, Change Cipher Spec, Hello Request, Hello Request
...	4...	::1	::1	TLSv...	699	Application Data
...	4...	::1	::1	TLSv...	350	New Session Ticket, Change Cipher Spec, Encrypted Handshake Message
...	4...	::1	::1	TLSv...	993	Application Data, Application Data
...	4...	::1	::1	TLSv...	611	Application Data
...	4...	::1	::1	TLSv...	714	Application Data, Application Data

Secure Sockets Layer						
▼ TLSv1.2 Record Layer: Application Data Protocol: http						
Content Type: Application Data (23)						
Version: TLS 1.2 (0x0303)						
Length: 618						
Encrypted Application Data: 0000000000000018e3ed6fe83a1c57e093f5c12ef676cab...						
00e0	12	29	5d	e0	7a	62 90 74 c7 0c 08 46 f0 8c 8f 03 .).zb.t ...F...
00f0	3b	6a	72	21	c0	af c1 f1 f9 8c 5d 36 c9 9f 12 bc ;jr!....]6....
0100	87	8b	2c	8e	e0	6d c0 9d 1f 48 ae 37 8e 2d 5c 33 ,.,m.. .H.7.-\3
0110	51	e5	56	04	12	1a 4b 9a 30 97 6f 8c 58 a5 1c b2 Q.V...K. 0.o.X...
0120	16	b9	36	a2	d8	78 a6 10 f5 6c 6f 27 f5 43 9d 16 ..6..x.. 'o'.C..
0130	31	e8	27	e5	dc	06 ea a3 b4 de 02 17 57 18 ff 29 1.'..... .W..)
0140	0f	68	5d	a7	c1	c6 c8 a5 07 50 36 5b e2 ce a8 1a .h]..... .P6[....
0150	04	02	ea	f6	5e	c2 c8 db 43 22 24 95 81 51 6f 3e^... C"\$..Qo>
0160	6b	bd	68	c5	16	66 cc 97 36 e1 86 75 86 ed 42 bd k.h..f.. 6..u..B.
0170	35	1e	f7	e6	d1	97 49 2f 82 94 2b c7 da f8 47 24 5.....I/ ...+...G\$
0180	3f	ed	24	e1	c0	47 95 47 f5 da ae 76 59 a3 44 32 ?.\$.G.G ...vY.D2
0190	9b	46	44	79	35	8e 73 9b 64 23 6b 60 fa 6f de 24 .FDy5.s. d#k`.o.\$
01a0	6f	43	16	43	92	dd 0f 49 de 4e 84 b1 28 97 98 1c oC.C...I .N..(...
01b0	6b	5f	ee	f3	71	82 21 11 19 d6 4f 63 33 cc da 63 k_.q.!. ..0c3..c
01c0	b0	55	a6	62	3a	e9 2a 0c d8 89 5a e9 d6 f5 75 d8 .U.b:*. ..Z...u.
01d0	c1	06	e7	e1	2d	e3 17 41 dc a7 97 18 32 87 b5 14-..A2...
01e0	68	7f	1f	b8	02	3d 5f 4d 64 e6 de 5d 94 29 0d 56 h....=_M d..].)V
01f0	f4	9a	78	aa	7a	2d 2c d3 24 13 54 98 75 b9 dc f1 ...x.z-,. \$.T.u...
0200	3c	d1	50	fd	61	51 46 90 bf f9 84 91 cb b3 36 fa <.P.aQF.6.
0210	61	25	d4	ee	cb	fd 95 1e 99 0a 3e d7 57 b8 69 89 a%...... ..>.W.i.
0220	7f	3c	57	cb	de	04 5d a5 cc 90 7d 5c ba f5 57 a2 .<W...]. ..}\.W.

Figure 25. Client / Server communication encrypted.

On figure 25, all communication is entirely encrypted between Client and Server and we cannot have any information about the type of communication. By this demonstration, we prove that HTTPS is working properly and ensure that data flow is encrypted against MITM attack.

We must remember that we are using a self-signed certificate. Thus, MITM attack is still affordable but for a real distribution, we would use a CA signed certificate (e.g. Letsencrypt) and ensure that the certificate cannot be spoof.

7 Discussion

On this chapter, we are going to discuss about technologies used through the product development, giving a clear explanation of each technology and its positive and/or negative points and ways of improvement.

7.1 MEAN stack

The developed product is a MEAN stack application, similar to the LAMP stack (Linux OS, Apache Web Server, MySQL & PHP), introducing JavaScript language, which used to be a front-end side language, as a back-end side language. MEAN tool is then providing web-services made of a homogenous programming language with a high scalability, matching perfectly for an IoT solution which combines electronic hardware devices with Internet.

Representational State Transfer (REST) uses HTTP operations POST, GET, PUT & DELETE to interact with Create, Read, Update, Delete (CRUD) database operations. Thus, a simple API, per example, can be used to create a new connected object into the database, due to the scalability provided by MongoDB, by any authenticated user.

Another undeniable advantage is the use of JSON format. Besides JSON is easier for a human reader to understand, computers and libraries easily parse it which makes the communication between two different programs or languages easier to transfer. (Poulter, Johnston, & Cox, 2016)

7.2 JWT

A JWT represent claims that are transferred between two parties through the Authorization header. These claims are a JSON objects encoded in JSON Web Signature (JWS) or JSON Web Encryption (JWE).

Javascript Object Signing and Encryption (JOSE) framework provides specifications within Headers to describe the cryptographic operations applied to the JWT (JWS or JWE).

A JWT is composed of three parts split by dots and seeded in the following order:

1. JOSE Headers giving the token type and algorithm.
2. Payload which contains the data of the token.
3. Verified signature or encryption.

HEADER: ALGORITHM & TOKEN TYPE
<pre>{ "typ": "JWT", "alg": "HS256" }</pre>
PAYLOAD: DATA
<pre>{ "_id": "58bac6ef01995d173392f4f2", "name": "test", "password": "\$2a\$10\$3/ug0sKKihYW5zM2rkHGr.NqWckf5QVeVpbhRI1IoF/ayj0dmaSXq", "__v": 0, "settings": { "customNumbers": [] } }</pre>
VERIFY SIGNATURE
<pre>HMACSHA256(base64UrlEncode(header) + "." + base64UrlEncode(payload), the-secret) <input type="checkbox"/> secret base64 encoded</pre>

Figure 26. Decoded token and its three sections.

On figure 26, we decoded the token received in chapter 6.1. The two first sections (JOSE Headers and Payload claims) are readable as they are simply Base64URL encoded.

Firstly, the JOSE Headers section defines the type and algorithm information. There are two algorithms available, HS256 which is a synchronous encryption algorithm which can be replaced by RS256 algorithm which is asynchronous (public and private key). An example of unsecure JWT would be to have a JSON Header where the attribute *alg* is *none*, which would mean that the third section doesn't require any verification.

Secondly, the data section which is our JWT Claims Set and has three classes of Claim Names: Registered Claim Names (RCN), Public Claim Names (PubCN), and Private Claim Names (PrivCN). RCN has defined claim names (e.g. *iss (Issuer)*, *sub (Subject)*, *exp (Expiration Time)*, *etc...*) and none of them is mandatory to implement. The PubCN is define by the JWT producer and has to be registered. Then, PrivCN are define by both, producer and consumer, and may agree on those to avoid conflicts. On figure 26, we are using Public Claim Names define on server side.

The last section is for verifying the signature of the token by applying a JWS or JWE steps for validating the JWT. On this section, we have to apply the cryptographic algorithm defined on JOSE Header (with the secret key) to ensure of the authenticity of the token. (Melorose, Perroy, & Careas, 2015)

7.3 Motion

Motion is a software offering simple motion detection with a camera and able to stream it through FFMPEG and over a server. The camera used for the project was a Raspicam with 5MP⁷ which is a low quality for taking pictures. Using a higher resolution camera would provide a higher quality of the events in order to do deep learning and create a neural network.

Unfortunately, even with a better camera, the software has some limitations and does not offer all the features required for the future development of the project such as:

- Detection of only human beings and not all kind of motions on camera.
- Streaming directly over internet through our server.
- Motion JPEG does not provide a fluid video and does latency on the records.

As the software is open source, some of these features can be fixed by changing the code directly but it would have required a lot of time to customize the software. Thus, we decided to create bash scripts in order to replace the lack of features.

A solution to replace Motion is to use OpenFace, which is an open source face recognition with deep neural networks. It uses dlib or OpenCV for detecting faces, then transform the face for the neural network to recognizing a human in real-time. This solution will be applied to the product in the future development but may require a more powerful

⁷ Megapixels is an unit of measurement for digital resolution.

computer as real-time face recognition needs to detect a face, crop it, and extract the features to be analyzed into the neural network.

7.4 Internet of Things

Implementing an IoT solution can be done as a Do It Yourself (DIY) project. The developed product can then be used to control any kind of electronic component (e.g. steam detection, flood detection, motion detectors, indoor/outdoor temperature, etc...).

Although implementing your own IoT solution, as a DIY project, may require carefulness about the external components used and connected to any kind of computer device to the existing electrical wiring. Thus, we are going to discuss about the mathematics required to calculate a circuit and know if there is no risk for the testing phase. As mentioned in chapter 5.5.1, we only worked with low voltage components. As these components are cheap, they can be destroyed for a learning purpose (e.g. testing 5v over a LED results in a burned diode).


The information discussed herein below is not sufficient to implement high voltage circuits (e.g. AC 220v). In all cases, advanced electrical knowledge is required to handle high voltage things which can be combined to a computer. A wrong manipulation of an electrical circuit can result in an uncontrolled fire, the destruction of all electronic components or death. (Vujovi & Maksimovi, 2015)




7.5 An introduction to electronic

As discussed previously, any implementation of external electrical components requires knowledge about how electricity behaves in order to avoid any wrong manipulation. In this section, we are not going to go deep in electricity understanding but only on the essential for a small implementation of electronic components with a computer.

7.5.1 Electronic symbols

As for UML, electronics have a design pattern in order to understand a circuit without having to implement it. In the implementation of the product, we are showing the symbols required to do a simple implementation of an IoT solution linked to a computer.

	Direct Current (DC): it is a stable current, electrons are moving in the same direction and its unit is voltage V .
---	---

	<p>Resistor: it is used to control the intensity of a circuit and its unit is Ohm Ω.</p>
	<p>Relay: it serves to control higher voltages and it works with an electromagnetic system which changes the state of an internal wire.</p>
	<p>LED: it is composed of an anode (positive side +) and cathode (negative side -). The voltage required by a LED differs from its colour and the amps (I) are usually around 0.02A (2mA).</p>

7.5.2 Ohm's law

Ohm's law is a simple formula which will be used to calculate the resistance and intensity of a circuit. Other formulas, such as for calculating Watts, can be used. However, to obtain a simplified IoT solution, Ohm's law is sufficient.

$$I = \frac{V}{R} \text{ or } R = \frac{V}{I} \text{ or } V = RI$$

$V = \text{Voltage}$ and has to be known to implement a circuit. All examples hereinafter will be on a circuit with a DC of 3.3v.

$R = \text{Resistance}$ is the measure of the difficulty to pass an electric current through a conductor, we can know it but also calculate it to control the amps going through our circuit (e.g. we can light a LED of 2v and 0.02amps but we need a $R = \frac{3.3V}{0.02A} \rightarrow R = 165\Omega$. We ensure that the LED will not be destroyed).

$I = \text{Intensity}$ is expressed in amps (A) and is the number of electrons going through the circuit in a determined time. Amps vary to the sum of resistances used (e.g. on a series circuit with two resistances of 220Ω , we have an $I = \frac{3.3}{220+220} \rightarrow I = 7.5mA$).

7.5.3 Series and parallel circuits

Circuits can be mounted in two different ways:

- *series circuit*: $R_{total} = R_1 + R_2 + \dots + R_n$ & $I = I_1 = I_2 = \dots = I_n$
- *parallel circuit*: $R_{total} = \frac{1}{\frac{1}{R_1} + \dots + \frac{1}{R_n}}$ & $I = I_1 + I_2 + I_n$

Then, in a series circuit, the R_{total} is equal to the sum of all resistances in the circuit and the I is the same for all circuit. In a parallel circuit, I is equal to the sum of all intensities of the parallel circuit and R_{total} is calculated for each parallel circuit and summed to the resistances in series of the circuit.

In our implementation, we only used series circuit to have an easier and safely implementation of the IoT solution.

7.6 Conclusion

Developing a MEAN application is coherent with nowadays requirements of customers and product qualities. Internet is constantly changing and growing through the years. Thus, reliability, security scalability, maintainability, portability of a product is crucial.

A MEAN stack is then ready to face to these unknown evolutions, through the development of APIs easy to update and maintain, a NoSQL database giving a high scalability to the product, and a growing community reporting issues to update security leaks of these new technologies.

The development of this product has given me a significant confidence on web development, by learning Express for Node.js and an interest for ongoing technologies such as Angular2 and React frameworks.

Moreover, IoT and video encoding introduced me to Machine Learning and how to implement an artificial intelligence (AI) able to learn through the days its owners and be more efficient in understanding their indoor behaviour. Google released, in 2016, TensorFlow, an open-source software library for Machine Learning, giving the opportunity for all community to create their own AI without having a deep understanding of mathematics applied.

In conclusion, this product, on its current state, can be implement in an indoor environment as a security system and to control objects with two states (ON/OFF) and PIR sensors. As for any product, it may require updates that will be done in the next month's such as access rights, an AI to understand owners' behaviour and facial recognition, and many unknown others in order to fulfil customers' requirements.

References

- Ansari, A. N., Sedky, M., Sharma, N., & Tyagil, A. (2015). An Internet of Things Approach for Motion Detection using Raspberry Pi. *Intelligent Computing and Internet of Things (ICIT), 2014 International Conference On. 17-18 Jan. 2015. Harbin, China*, 131–134. <https://doi.org/10.1109/ICAIOT.2015.7111554>
- Haviv, A. Q. (2014). *MEAN Web Development. eBook* (Vol. 1).
- Howard, M. (n.d.). Man-in-the-Middle Attack to the HTTPS Protocol.
- Melrose, J., Perroy, R., & Careas, S. (2015). JSON Web Token. *Statewide Agricultural Land Use Baseline 2015, 1*, 1–30.
- Poulter, A. J., Johnston, S. J., & Cox, S. J. (2016). Using the MEAN stack to implement a RESTful service for an Internet of Things application. *IEEE World Forum on Internet of Things, WF-IoT 2015 - Proceedings*, 280–285. <https://doi.org/10.1109/WF-IoT.2015.7389066>
- Vujovi, V., & Maksimovi, M. (2015). Raspberry Pi as a Sensor Web node for home automation. *Computers and Electrical Engineering*, *44*, 153–171. <https://doi.org/10.1016/j.compeleceng.2015.01.019>

Appendices