

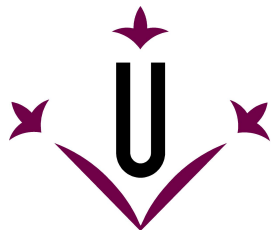
LIGHTS

A thesis done by

RAMON GILABERT LLOP

Electronic, Automatic and Industrial Engineering

2015 - 2016



Universitat de Lleida
Escola Politècnica Superior

Fernando Guirado as director

This project is, without any doubt, the biggest side project I have ever had, so many technologies and innovation are involved that I have to name some of the people that has helped me go through the process of making it a reality; from the first time when I thought that I could merge two worlds, up to the point in the reviewing and tinkering phase. Fernando, Tim, Josep, Norman, Laia, Ramon and many many more; this goes to you and to all the people that, as me, are ambitious enough to think they can, in a small way, change how things normally work and truly make this world a bit of a better place.

INDEX

1. Disclaimer.....	1
1.1. The document.....	1
1.2. Brief.....	1
1.3. References	2
2. Introduction.....	3
2.1. History the internet of things.....	3
2.2. References	6
3. Market studies and comparison.....	7
3.1. Hardware related products and flaws	7
3.2. Software related products and flaws	8
3.3. References	9
4. Objectives and encompass	10
4.1. Goals from Lights.....	10
4.2. Encompass.....	11
4.3. Planning.....	11
4.3.1. Explanation	11
4.3.2. Gantt diagram	14
5. The making of.....	15
5.1. Architecture	15
5.1.1. The platform	16
5.1.2. The product	17
5.2. Development brief	18
5.3. The generic platform into technical detail.....	20
5.3.1. Lights Backend	20
5.3.1.1. Introduction with database	20
5.3.1.2. HTTP methods	21
5.3.1.3. Web sockets.....	22
5.3.2. Lights Berry	22
5.3.2.1. Introduction with database	23

5.3.2.2. Main functionality	24
5.3.2.3. Sockets.....	25
5.3.2.4. Bluetooth communication	25
5.4. The product into technical detail	25
5.4.1. Lights Backend adaptations.....	26
5.4.1.1. Database	26
5.4.1.2. HTTP methods	26
5.4.1.3. Sockets.....	27
5.4.2. Lights Berry adaptations.....	28
5.4.2.1. Database	28
5.4.2.2. Sockets.....	28
5.4.2.3. Bluetooth communication	28
5.4.3. Lights Duino	29
5.4.3.1. Bluetooth in the Arduino	30
5.4.3.2. Handling of the light.....	32
5.4.4. Lights app.....	32
5.4.4.1. Flow of the app	34
5.4.4.2. Inner operations of the app	36
5.4.5. Industrial design.....	37
5.4.5.1. Design of the hub.....	37
5.4.5.2. Design of the light.....	38
5.5. References	40
6. Tests and results.....	41
6.1. Error prone situations	42
6.1.1. Bluetooth tests and results.....	43
6.1.2. Internet tests and results.....	46
6.1.3. Stress tests	47
6.2. Process test.....	49
7. Budget and engineering.....	50
7.1. General overview of Lights.....	51
7.1.1. Creative and architectural phase.....	51
7.1.2. Development, testing and engineering phase	52
7.1.3. Design, prototype, tinkering	54

7.2. Total cost estimations	54
7.2.1. One time costs	54
7.2.2. Per light cost	55
7.2.3. Mass cost	57
7.3. Market study and general public	58
7.4. Final cost estimation	59
7.5. References	59
8. Motivation and conclusions	61
8.1. Motivation	61
8.2. Why Lights	61
8.3. Conclusions	62
9. General reference list	64
9.1. Bibliography	64
9.2. Webliography	64

1. DISCLAIMER

1.1. THE DOCUMENT

As a full disclosure before starting the document, this thesis follows different guidelines on how to make the document, how to structure it and how to make the presentation out of it. Concretely, it conforms to the UNE 50135^[1], the guidelines of the University of Lleida and the guidelines of the University of North Caroline^[2].

The font chosen to do this document is a serif font with a size of 12 points to ensure readability in all the distributed copies.

1.2. BRIEF

There has been a lot of technologies involved in this project which makes it hard to cover them all, to simplify the readability of the project, this section starts by introducing some of its key components.

There are two main concepts within this project codenamed as **Lights**; a platform, which is a forum for other devices to connect to, and a product, which represents the final and central efforts within a visual gadget such as a smart bulb that is able to perform multiple operations listening to commands coming from different places, normally with a starting point in an iOS app although prepared, together with the first platform, to be hooked into multiple solutions like web, Android, Windows Phone etc.

Inside Lights there are multiple systems, services and components. The most important ones, which create the structure of the platform are two servers, one micro-controller and one app.

The codenames of those systems are the following:

- Lights-Backend (for the cloud server)^[3].
- Lights-Berry (for the Raspberry Pi server)^[4].
- Lights-Duino (for the micro-controller (Arduino) code)^[5].
- Lights (for the native iOS app)^[6].

There are multiple other components, scripts and snippets of code created out of Lights that are mentioned in section 2 of the annex II.

1.3. REFERENCES

[1]. UC3M. How to write a scientific document [2016, May 14]. Retrieved from:

http://docubib.uc3m.es/CURSOS/Documentos_cientificos/Normas%20y%20directrices/UNE_50135=ISO%205966.pdf

[2]. UNC. Formatting guidelines in a thesis [2016, May 14]. Retrieved from: <http://gradschool.unc.edu/academics/thesis-diss/guide/format.html>

[3]. GitHub. Lights Backend [2016, May 15]. Retrieved from: <https://github.com/RamonGilabert/Lights-Backend>

[4]. GitHub. Lights Berry [2016, May 15]. Retrieved from: <https://github.com/RamonGilabert/Lights-Berry>

[5]. GitHub. Lights Duino [2016, May 15]. Retrieved from: <https://github.com/RamonGilabert/Lights-Duino>

[6]. GitHub. Lights [2016, May 15]. Retrieved from: <https://github.com/RamonGilabert/Lights>

2. INTRODUCTION

Lights comes and belongs to different industries and engineering fields, considered itself as a component of the new era of the Internet of Things, it also is a representation in the paradigm of advanced home automation.

2.1. HISTORY THE INTERNET OF THINGS

There is a starting point to every revolution and, with every iteration within technology, innovation gets bigger and better. The starting point for this introduction takes place in the XIX century in Switzerland with a well known concept nowadays, home automation.

The union of the words home and automation gives meaning to the portmanteau as the use of automatic or remote means to regulate, create or change technology systems in homes, houses or places. There are various examples of home automation, such as automatic curtains, smart thermostats, etc.

The concept per such begins in 1885 when the Swiss engineer Albert Butz^[1] invented a thermostat that would automatically regulate heating systems. Not too long after that, automation devices started to become an expensive luxury gadget for society, having a huge impact in the inventors and engineers at the end of the XIX century. People like Mark Honeywell or Nikola Tesla patented multiple devices such as remote controls or more advanced heating system regulators that could be controlled remotely among others; the problem with that was the huge price of it, which made it something not as transcendental as such inventors wanted it to be.

As a parallel story, the revolution of computers from the 1970s started making technology cheaper and more accessible to a normal person life, while a computer was a luxury in homes, some of the best schools around the countries had one of those machines for its students to start evolving their curiosity around the industry.

That fact, started to create a small tendency in young people to be interested in the technology field and, those gadgets that once seemed inaccessible and were thought only for collectivists and enthusiasts around the world, would soon become something everybody would need.

Such revolution made software accessible for everybody to know and learn. At the end of the 80s, geeks from the best houses started to buy computers and kits to start installing components into it, development started to be a thing within normal people and not something left to engineers, and parallel to this, the biggest technology companies were created or entered in a huge bubble that would catapult its fame, companies such as Apple, IBM, Microsoft or Dell among others started to make the personal computer as a box that people would just carry around and not something huge and expensive.

After about 30 years, in the 2000s, the web bubble started, the companies mentioned before would be in top of its business and everybody would have a computer at their homes. The problem was, a computer was still luxury and the problem still remained, hardware was not that accessible to people as software started to be in the 70s.

It has not been until the second decade of the XXI century when, with the increasing popularity of technology and research combined with the increasing amount of engineers among others that home automation, and specially cheaper hardware has gained track again. Smartphones, tablets, convertibles and computers are something in a day to day life in everybody's life; that fact opens a world for engineers that is insanely huge; the capacity to get to a website or to discover a product in just one click, control a car within a phone, open the doors 300 miles away from home within your watch, connect to people around the world with milliseconds of delay, etc. is something that was unimaginable just 15 years ago.

The fact that the iPhone was introduced 9 years ago has made technology available to everybody; 2016 is the year of artificial intelligence, smart-things, connected devices, virtual reality, etc. and, putting a lot of attention into the smart part of the sentence and going back to home automation and the ability or the seeking of controlling gadgets at home, it can be said that there is a relation created out of the abstraction of the Internet of Things. Such concept seeks for connected smart devices that talk to each other in order to make our day to day easier; smart objects that know and get to know every personal user by being used and are able to communicate to them at the right time, in the right way.

The future is unknown but, the fact that an ABI research shows that only in 2012, 1.5 million^[2] devices were installed across the United States that were considered home automation devices, with a projection of 6.4 billion gadgets by 2020^[3], presents a huge revolution for engineers, showing that this revolution is in no peak, this is the starting point of it, the beginning of the path. That also comes with more technology companies in Silicon Valley and everywhere around the world pushing for those devices, Nest^[4], with its smart thermostat, Philips with its Hue smart bulb or in the other hand Apple with HomeKit^[5], a platform to easy connect everything around home; lights, curtains, heating systems, etc. with a new application coming in iOS 10 within it, called Home^[6].

In a world where time is one of the things that are most appreciated, smartness, easiness and fast delivery of information within a fast deployment of it makes the relevance of any platform that can deliver that to increase its value exponentially as time gets more precious day after day.

In the next few years home automation will disappear letting the Internet of Things take over it, this natural evolution will create and creates smart connected devices that achieve the automation that the first one sought; having everything connected to get that information as fast as it can be and whenever is best for us. Every day there

are new products in the field, cheaper hardware means cheaper devices and more evolved technology means easiness to use those devices with more users within it.

2.2. REFERENCES

- [1]. Honeywell. Honeywell history and inventions [2016, May 15]. Retrieved from: <http://twincities.honeywell.com/honeywell-history-and-minnesota-heritage/>
- [2]. ABI research. 1.5 Million Home Automation Systems Installed in the US This Year [2016, May 15]. Retrieved from: <https://www.abiresearch.com/press/15-million-home-automation-systems-installed-in-th/>
- [3]. Gartner. 6.4 Billion Connected "Things" Will Be in Use in 2016, Up 30 Percent From 2015 [2016, May 17]. Retrieved from: <http://www.gartner.com/newsroom/id/3165317>
- [4]. Nest (Alphabet). A smart thermostat [2016, May 17]. Retrieved from: <https://nest.com>
- [5]. HomeKit (Apple). A platform for the Internet of Things [2016, May 17]. Retrieved from: <http://www.apple.com/ios/homekit/>
- [6]. Home (Apple). An app to interact with the internet of things devices [2016, June 19]. Retrieved from: <http://www.apple.com/newsroom/2016/06/apple-previews-ios-10-biggest-ios-release-ever.html>

3. MARKET STUDIES AND COMPARISON

Before starting the phase of development, there is a period of time searching for alternatives in the market, projects already done, open sourced platforms, do it yourself alternatives, etc. As stated in the introduction, one of the most famous smart lamps and internet of things exponents that exist in the market nowadays is Hue by Philips^[1]. Said lamp has been a big inspiration while developing Lights as a product, whereas Apple HomeKit, a smart framework that gives software to unify hardware components, has been a huge mirror when developing Lights as a platform.

3.1. HARDWARE RELATED PRODUCTS AND FLAWS

Hue is a smart lamp that works more or less the same way Lights does, the only difference is that, while the price of Hue is around 150\$ a pack (3 bulbs and a hub), the price of Lights, as stated in section 7, is much cheaper at around 80\$ per pack (3 bulbs and a hub) in mass production.

Looking at the public market that Hue has as of 2016, the biggest challenge that raises is that only a geek market buys those kind of gadgets, thus, it makes the public of it smaller and tighter. In the particular case of Hue, the design of the lamp is very futuristic, white and simple for people to think forward when having their product and, while this is good for technology fans, Lights is intended to be something else. The same occurs with Nest, a thermostat that is so futuristic and, probably ahead of time that only geek people know about.

A part from Philips, no other major alternatives exist while looking the same market space within the product, some crowdsourcing campaigns have raised but represent just what the product of Lights is and not the platform and, while there are lots of tutorials to do similar things with the technology; controlling an Arduino via Bluetooth, etc.^{[2][3]} There is nothing quite similar as the structure created for Lights

and explained in the brief in section 1.2 where everything is thought for the end user to be just simple.

One of the biggest problems with the do it your self alternatives or close to market devices is that they talk about connecting a device via Bluetooth right away to the micro controller without the need to talk to a hub or to a server first. That fact kills the idea of scalability, easiness and platform creation; scalability for an obvious reason since there is the need of a hub to control the end light separately, with a server in the cloud to have basically control over all the devices around; easiness because it needs special configuration regulated and different in every case having a lot of error prone situations as what happens if it disconnects, the bluetooth fails, etc.

Even though the market is growing as of 2016 in terms of home automation or internet of things; smart bulbs, lights, lamps and devices connected have not entered yet into a bubble, making them something new for users to try still. Such fact creates a really open market that is still let to interpretation with so many possibilities to make it grow and the possibility to do lots of studies around it.

3.2. SOFTWARE RELATED PRODUCTS AND FLAWS

Apple HomeKit is the mirror when looking directly at Lights. While there is no such thing as some platform that combines both hardware and software, this is the best alternative in the software field. HomeKit is a framework that gives the ability for people to control all the devices around their houses within the phone and with just a couple of clicks. For that, Apple has created an API for other providers and manufacturers of home automation devices to enter the internet of things era in an easy way, unifying then the concept and making it generic for people to use.

While this cannot be topped because the boundaries of Lights are now within its product, that does not mean that scalability is something that has not been in the

mind of the project since day one. That is why, at this point, HomeKit is the mirror Lights is looking at to.

Other than that, there is no big provider of an API that can solve what HomeKit does. Every fabricant of home automated devices has a small client in order for the user to control the light, the thermostat or the smart gadget in place, but no big platform is out there other than Apples’.

3.3. REFERENCES

[1]. Hue (Philips). A smart lamp and bulb connected with an intermediary hub [2016, May 17]. Retrieved from: <http://www2.meethue.com/nl-no/>

[2]. Instructables. NodeJS and web-sockets tutorial [2016, May 20]. Retrieved from: <http://www.instructables.com/id/Easy-NodeJS-WebSockets-LED-Controller-for-Raspberr/>

[3]. Servicelab. Control an Arduino with NodeJS over Bluetooth [2016, May 20]. Retrieved from: <http://servicelab.org/2012/12/12/wirelessly-control-your-arduino-with-nodejs-over-bluetooth/>

4. OBJECTIVES AND ENCOMPASS

This section provides the goals to achieve at the end of Lights as well as the encompass of the project; such section serves as the introduction of the development showing the planning of it.

4.1. GOALS FROM LIGHTS

The objectives appear in a list and later on are explained.

- Research in how to merge electronics and software with exceptional design.
- Accomplish a prototype that can be used anywhere and from anywhere in the world.
- Add easiness in the installation as well as in the usage of the platform and the product.
- Accomplish a great peer to peer connectivity in Bluetooth that is robust enough to work with a maximum of 1% of error.
- Create a scalable system that can be ported into mass production involving multiple hardware and “internet of things” solutions.
- Get a prototype of the product that can be ported to start a crowdsourcing campaign.

The main objective, shown in the first line, is to merge the worlds of electronics and computer science creating something that uses both, electronics to control a light, change the color of it, etc. and a development open enough to build the skeleton of the platform in which Lights is built upon, letting the possibility for other products to enter into such environment too.

As stated in the previous list, Lights pretends to be one of the first kind of home automated devices that are for the general public even though it is technologically advanced enough for only marketing geek people; with that, it has to be a product that is robust enough to be scalable into mass production and easy enough for people to use it right away. Even though this might seem like a small thing, one of the biggest difficulties that appear while developing a platform, a product or anything that is user based, is thinking on how the end user will make use of it; covering all the cases is difficult and having no errors is pretty much impossible, that is why making it easy enough to engage the end user with it is a huge objective in the list.

4.2. ENCOMPASS

The encompass of the project consists in creating a platform with four different layers of computing that include a product within them. Such platform can grow as much as porting it to a crowdsourcing campaign, letting it ready enough for other people to contribute to it, making it bigger and better overtime.

4.3. PLANNING

4.3.1. EXPLANATION

There is a lot to do in the process of making Lights, lots of technologies mean lots of industries involved, thus, all the concepts are separated in different blocks that represent different fields.

I. Creative phase

The creative phase is the one in which the process of pitching different ideas begins, what do to, how to do it, etc. The process starts by making a brainstorm of ideas as well as commenting such ideas with different people to get feedback on it. That

phase takes place in October and lasts for a month. The process ends when, after some research and questions, there is an agreement on what to do. The idea with Lights is to make something that is fun, serious and meaningful enough to be able to compete in the market of the internet of things.

II. Initial planning phase

The initial phase starts when, after comparing opinions and gathering feedback of other people about the result of the creative phase, the idea is ready to be built; such phase starts by planning what is needed for the project as well as what is the definition of it. In this case there is the need of two backends written in JavaScript in order for them to be fast enough, an app and a client. For that there is the need to put some padding in the plan in order to learn the technologies, build the prototype, etc.

Such phase finishes talking with the coach of the project, deciding and writing down in a more formal way what the project is supposed to be and how it is supposed to be structured.

III. Learning phase

With the basic architecture already planned in point 2, there is the need to learn the technologies required to build it, in this case different languages like JavaScript, the Raspberry Pi fundamentals, 3D design, etc.

In a field like development and design where everything moves in a huge velocity, the learning phase never ends, that is no different here since the learning process goes through all the project even though the formal part ends around January; starting after that the making of the project.

IV. Maker phase

After the learning phase, it makes sense to do the parts of the platform that do not require any design, those are the backends, the micro-controller code, etc. The architecture is already prepared in point 2 and has more add ons and revisions in this phase.

V. Design phase

With the development of the non designed parts done, there is the need to have the look and feel for the app and for the hardware of the project. It is important to notice the separation between developments; the reason for that is to put a bit of padding in the concepts to solidify the first part of development, entering in a more stronger way later on for the product. Such phase includes the first mocks of the app and the industrial design of the lamp, as well as some design parts for the report, documents and other files attached to the project.

VI. Final development

This phase includes the last part of the designed pieces of the platform and the product. Implementing the app as per such and the prototype of the design into the real components.

In this phase there is a going back and forth with phase IV, since there is a lot of non intentional testing of the non designed product.

This phase takes place during 1 month and a half, without including tests rather than the actual formal ones done in order for instance for the app to work.

VII. Report, test, examples, finishing

This is the last part of the process, also called quality part. This includes documenting the project, testing the prototype, building the examples, doing some more design for the document, etc.

4.3.2. GANTT DIAGRAM

In a more technical way, a summarize of the different phases of the project seen in the previous section can be represented as something like the following diagram:

Table 1. Gantt diagram with the phases, the structure and the timing of Lights.

	October	November	December	January	February	March	April	May	June	July
I										
II										
III										
IV										
V										
VI										
VII										

Some of the tasks overlap as the learning and making phase since one helps and can complement the other.

5. THE MAKING OF

This section presents Lights into technology detail; why and which are the decisions that have been made and an in depth look into the software involved. This process is separated into the parts in which the platform is divided, later on presenting the product as per one simple client of the platform. In such last subsection, the parts that need modification in order for the product to work are also shown.

To begin with the explanation of the project, the architecture shows the platform and the product combined at first, deconstructing them into separate components later on.

5.1. ARCHITECTURE

Lights consists in a generic platform and a product. Such product is hooked into a tweaked version of such generic platform in order for it to function.

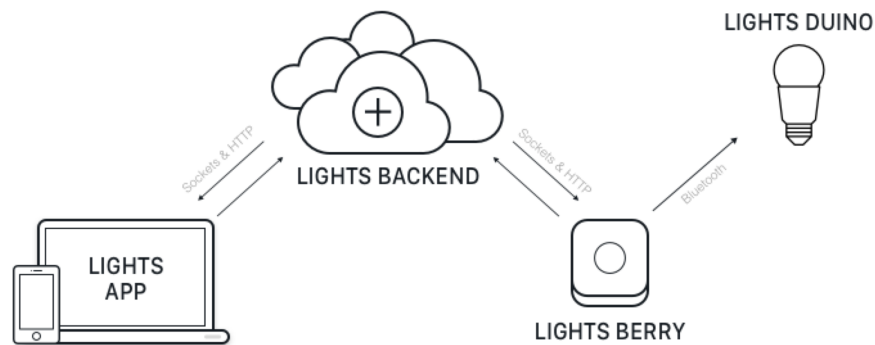


Figure 1. General architecture showing every component.

Figure 1 can be deconstructed into two parts as already stated. The best way to understand such parts is by putting an example of a thermostat and comparing it with the case that it is being covered in this section.

The splitting needs to be done in order for other products to enter or, in another way, be part of a sharing trade between components. Continuing with the example of the

thermostat, the concepts that could be easily shared are Lights Backend and Lights Berry; tweaking the database in Lights Backend and changing some parameters in Lights Berry, Lights becomes a completely different product by just adding another micro-controller with bluetooth on it and changing a little bit the app.

The conclusion is then that the components Lights Backend and Berry create the platform whereas the micro-controller and the iOS app, alongside the platform, create the product.

5.1.1. THE PLATFORM

The platform is, as said, the generic part that can be extracted and is independent enough to be able to hold different products. That contains:

- A server hosted in the cloud that incorporates all the information of all the smart objects that are used around.
- A server running in a Raspberry Pi that is always on and listening to the commands sent by the first backend. It performs basic operations like, parsing, saving in the internal databases or connect to the actual products and clients via Bluetooth.

Inside every component there are multiple scripts to automate concepts, builds and processes.

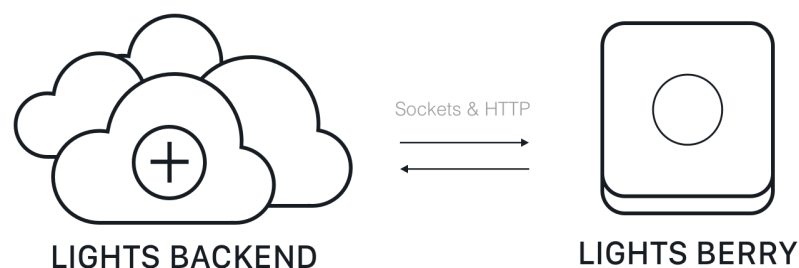


Figure 2. Diagram of the architecture of the platform.

The representation in figure 2 shows the connection between the backends of the platform, displaying the way they talk to each other and the communication they produce. Each component is explained separately and into detail in its correspondent section.

5.1.2. THE PRODUCT

The structure of Lights as a product consists in:

- The tweaked platform explained in section 5.3 with its extended functionality.
- A micro-controller, in this case an Arduino, that contains a light and a Bluetooth shield to be able to listen to the commands from the platform.
- A native iOS app that is able to send commands via web sockets.

Inside every component there are multiple scripts to automate concepts and processes as the platform does too.

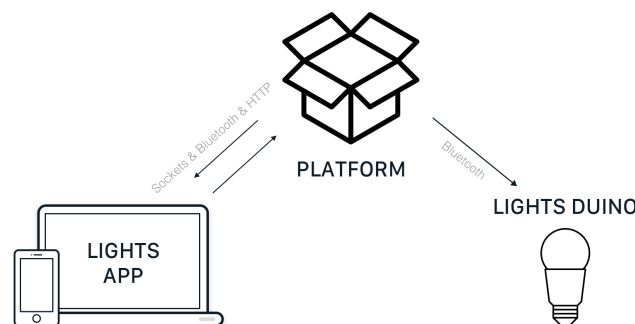


Figure 3. Diagram of the architecture of the product that communicates to the platform.

What the product achieves is basically to be as agnostic as possible from the platform, displaying and sending information into a funnel that can just send said information into the other end using what the platform is prepared to, sockets, bluetooth and HTTP.

Entering into more detail in terms of communication, the clients talk to the backend via web sockets or via HTTP, whereas the Raspberry Pi talks exclusively to its connected clients via Bluetooth, retrieving information from the other side of the platform via HTTP.

5.2. DEVELOPMENT BRIEF

Before explaining into a technical detail the solutions and the challenges, some of the questions that appear when thinking towards the architecture shall be answered. The obvious first one is why the need of two backends.

When performing the creation of a platform or any product, one of the first questions that raises is what to do with scalability. As porting the project to a mass production is one of the end goals and converting it to a platform is the main, having a solid fundament is necessary. There are then two possibilities to help solve this, either one, do just one system that lives in a Raspberry Pi, the called (Lights Berry) or do one that can hold multiple systems like the first one and can help controlling from the top other small components.

Developing ways, it would had been easier to do just one service that lives in the Raspberry Pi, however some problems would have had raised:

- The first one, the editing of the product when it is not connected since then, there is no hub or database all time live to save those changes.
- The second one, and the most important, the communication to the phone that could only be via Bluetooth and not internet, making it very limited and just due to a range of less than 30 meters from the actual object.

Those problems combined within the matter of not being able to make it as generic as possible for a platform, makes the this pattern not valuable to follow.

Finally in that matter, there is a side note that is important enough to note, displaying some of the first technical difficulties of Lights; while the connection from the phone to the product in terms of editing works with sockets, the connection to get all the lights or products, post them, etc. is done over HTTP. While sockets, as explained in section 1 of the annex II, break a bit the system of the communication over the internet, permitting the user to communicate to a port that is not opened, HTTP does need to have the port open in order for another third party to try to communicate with that server. That breaks one of the main rules when developing Lights, easiness. If the user needs to open a port or something similar to that to install the platform and use the product, it makes things really complicated and the adoption and learning curves too big for new users to get into it.

The way to solve such problems is to just make another backend, Lights Backend. That one has control over all the products. Lights Backend sends all the information of the lights to the clients over HTTP and is the hub for the sockets, emitting and receiving them.

Another question that raises is, why the need of a Bluetooth communication from the hub to the micro-controller, with a solution as easy as Lights Backend controlling a light in Lights Berry. The answer is basically that there is the need to be able to connect multiple products in one place, a light or multiple of them, a thermostat or multiple of them, etc. A part from that, Lights Berry has the limitation that is running in the Raspberry Pi, which needs PWM modulation to provide analog writing, making it more difficult to develop upon.

The idea of Lights is to make something open enough for other systems to be able to be plugged in into the backends to use, thus, there is the need to have Bluetooth and other technologies involved to connect multiple different micro controllers into it.

5.3. THE GENERIC PLATFORM INTO TECHNICAL DETAIL

5.3.1. LIGHTS BACKEND

Lights Backend is the first one of the components that are developed and conform to the platform. Lights Backend lives in the cloud, concretely in the platform *Heroku*, explained in section 1 of the annex II, and it is always accessible; that means that, even if no hub is connected to the current, the user is still able to edit the product status and see the effect the next time such product connects. That basically means that, an editing of a thermostat would be accessible and visible the next time such thermostat connects to the current.

Lights Backend consists in:

- Database written in PostgreSQL to store controllers and the products that such controllers hold.
- RESTful API to deliver the content of the database to the clients requesting for it.
- Socket emitter and receiver to handle the sockets from the different clients.

5.3.1.1. INTRODUCTION WITH DATABASE

The generic components of Lights Backend are basically the controllers. It also holds products which are specific to every use case and go tight to every controller.

I. Controllers

A controller is basically what the Raspberry Pi is in the platform, a hub. Lights Backend needs to have management over the controllers in order for it to create one to many relations with the products.

A part from that, there is a security reason why the controllers are stored in the Lights Backend; that one is the token. When a controller is formed, the backend creates a unique token that has to be sent in every request to modify a light; if that token is not the same as the controller's, the user is not able to edit the product. That prevents from other third parties to edit from an outside client over HTTP or sockets.

II. Products

The fact that it saves products means that the database can hold multiple generic values and can have multiple entities in them; an example of this is mentioned in section 8.2. A product could be a light, a thermostat, or anything that fits within the needs of the platform at that moment.

5.3.1.2. HTTP METHODS

As mentioned before, Lights Backend has two types of protocols to communicate to the devices, the first one is the normal HTTP method. Lights Backend is created to give a lean RESTful API that delivers JSON back to the different clients.

Although, for convenience, Lights Backend supports all the methods for an HTTP request, such as *GET*, *POST*, *PUT*, *PATCH* and *DELETE*; for each of the two entities, that is, *Controllers* and *Products*, that does not mean the end user or the Lights Berry or the client iOS app uses them. That is done basically to create a convenience and a protocol within all the API, making it leaner.

There is two headers required in Lights Backend; the convenience one called *Content-Type*, that defaults to *application/json*, which lets the backend understand what the return and the body type of the request should be, and the *controller_id* the request is referring to. With that, the backend is able to process such request. With a JSON response, the app is able to easily parse, store and edit all the key value components of it.

Finally, if the user wants to edit or delete a specific product, the request looks exactly the same as the previous one, with a convenience showing which light it needs to modify; that is done with the *ID* of that product and specified based on the convenience of the RESTful pattern as: */products/:id*.

5.3.1.3. WEB SOCKETS

The last form of communication to the backend is via web sockets. It is explained in section 1 of the annex II how a general socket works and in this case there is no difference. The backend is always able to connect to new sockets (clients); once a connection is established such backend is able to listen to messages from that socket. The messages are different depending on which client they are coming from.

The messages that the Lights Backend emits are basically for the client and the other server to update something, either the UI in the first case or the database in the second one.

Finally, it is important to note that the communication that the backend creates is a broadcaster one. Such backend sends messages to all the socket listeners, whereas the clients talking to the backend perform the same on a 1 on 1 communication.

5.3.2. LIGHTS BERRY

Lights Berry does not contain a RESTful API so it is not a typical HTTP backend even though it does similar operations as one, like controlling sockets, communicating via Bluetooth to its clients and controlling different products; the biggest difference with Lights Backend is that Lights Berry controls products instead of all the array of components that the first one manages.

Lights Berry consists in:

- Database written in PostgreSQL to store itself (the controller) and the different products it manages.
- A requester to the Lights Backend to get the last status of the products it is controlling.
- A socket emitter and capturer.
- A Bluetooth configuration to perform operations as a central module, connecting, sending and receiving information from others in a characteristic way.

5.3.2.1. INTRODUCTION WITH DATABASE

The database of this backend is similar to the previous one; it stores itself (a *controller*) with a limit of one row, and multiple *products* connected to the first one.

The only difference between this database and the one in Lights Backend is that the *products* and *controller* entities have another key called *address*. For *products*, this is the address of the bluetooth of the product that is controlling. For the *controller*, this is the address to the most recent phone, this last one is just for convenience and to keep a unified database across entities.

The purpose of this database is to store the current state of the controller and the products that is managing. The controller because is a requirement for more products to be created in the central database, and the products to store the status of them just in case there was a problem in the connection via Bluetooth.

Finally, is necessary to note that the database can be morphed into different needs and more entities can be added if needed for any specific product.

5.3.2.2. MAIN FUNCTIONALITY

Once the Raspberry Pi is connected to the current, it runs a script^[1] in order for the internal server to start running. This script, written in shell, is an init script that passes the appropriate parameters and starts a file with the logs of the server.

Once the server is on, it opens a green LED notifying the end user that the system is already usable.

The first thing such server does is to check the database and specially look if the *Controller* table is empty. If it is, it does a request to *POST* a new controller to the Lights Backend and saves it into the database. When this process is completed, it starts to search for a Bluetooth product and a Bluetooth phone indefinitely. Needs to be said that the Bluetooth searching for a phone is not done until the controller, and at least one product, are stored in the database. That is to always have, at least, one item in the app and not show an empty state as the first thing the user sees in the UI.

I. If it finds a product

If the server finds a product, it checks the peripheral and tries to connect with it retrying it until the connection is successful. This creates a connection that is not bound to be broken unless there is a problem with the current or something similar to it. The server tries to find other lights or products after the first connection is done.

Even if a problem raises, the server tries to reconnect until the light is connected again.

II. If it finds a phone

If the server finds a phone it sends the information of all the products that are stored in the local database to it, sending alongside the token of the controller in order for

the app to be able to get the information from Lights Backend, having present the security measures that the backends of the platform has.

5.3.2.3. SOCKETS

After a security check to see if the *controller_id* of the product corresponds to the one stored in the database it then updates, if it is, such product and notifies all the clients in order for them to update.

Inside such socket, information about the product is sent for a quick update of the local database and the online one.

5.3.2.4. BLUETOOTH COMMUNICATION

This is the most important part of Lights Berry, the Bluetooth communication. Once the peripherals are connected they can talk to each other via characteristics. The way it works is that every time there is a change in a product, Lights Berry receives a socket, creating from it characteristics to notify the clients connected to it.

Such characteristics can be configured in something the end product can need. In case of a light it can be an RGB value, in terms of a thermostat it can be a temperature. Such information needs to be as simple as possible since it is sent via buffers, which use data primitives to be represented.

5.4. THE PRODUCT INTO TECHNICAL DETAIL

The platform is where the product resides in. Changing the word product from now on for lights, this is the explanation of the making of the specifics.

Even though the platform is generic enough, it is configured to do what it needs to do, control an app and basically clients connected to it. Extracting the abstraction of the platform and its architecture and functionality, this section explains what is

added and what is needed in order for it to control a light with the help of other parts attached to it.

5.4.1. LIGHTS BACKEND ADAPTATIONS

This section inherits the explanations given in section 5.3.1 of this document and, as stated in the introduction, explains the minor improvements and changes in order to modify the platform to make it into a light connector.

5.4.1.1. DATABASE

I. Lights for Products

A light is the actual object and the crater of this explanation. In order for this identity to be created, it needs to have a *controller_id* in which to create a one to many relationship to. That means that, when a light is created a controller needs to have been created before.

A part from that, the light identity has different variables natural from a color object, those are, *status*, *intensity*, *red*, *green* and *blue*. As explained before, Lights uses the RGB pattern in order for the bulb to change its color in the LED in a more pleasant way.

5.4.1.2. HTTP METHODS

The main changes that the HTTP methods have is that they return *lights* instead of *products*. The headers are still the same and the requests look the same as they do in the platform.

As an example, when making a request to get the */lights* from the platform, the response is the one seen in the figure 4.

With the following JSON response as an example, the app is able to easily parse, store and edit all the key value components.

```
[
  {
    "id": 16,
    "created": "2016-04-07T00:00:00.000Z",
    "updated": "2016-04-07T00:00:00.000Z",
    "status": false,
    "intensity": 1,
    "red": 0,
    "green": 0.88,
    "blue": 1,
    "controller_id": 10,
    "address": "de:fd:bd:34:56:5d"
  }
]
```

Figure 4. JSON response example from the endpoint /lights.

5.4.1.3. SOCKETS

The present is the part where the most changes are made; not in how it works, because it does it in a generic way, but in the content of it. Web sockets work with messages; a set of keys retrieve a set of messages that, in this case Lights Backend can respond or do something about.

The messages that the backend supports for Lights as a product are 4: *ios-light*, *server-light*, *new-ios-light* and *delete-ios-light* as far as the codenames concern. The first two do the same thing, they create or edit a new light or an existing one depending on the starting point, that is, the iOS app or the Raspberry Pi server. Once the light has been edited, it notifies the other platform that there has been a change.

As an example, the server creates a light from the Raspberry Pi, then Lights Backend notifies the iOS app to update the UI based on the new information. In the other way around, when the iOS app updates a light, it notifies Lights Backend, which notifies the Raspberry Pi with another message saying that there has been a change.

The messages that Lights Backend emits are basically for the client and the other server to update something, either the UI in the first case or the database in the

second one. As said before, a socket can have a message, said message is the light itself in both cases.

To create a 1 on 1 communication with the client when the server broadcasts, the backend sends special parameters in the messages. The codenames of those are: *light-:id* and *ios-light-:id*. If the iOS app has the light's *id* specified in the message, it is able to parse it and use the content of such socket, always with a matching token.

5.4.2. LIGHTS BERRY ADAPTATIONS

Lights Berry inherits again the same configuration as it does Lights Backend from the platform, the only change is that it adapts the sockets and the database to get and represent what is needed in order for the light to change the color.

5.4.2.1. DATABASE

The product changes to confront the ability to save in the database the *color*, *intensity* and *status* of a light instead of the generic values of it. A part from that, the *address*, one to many relation, etc. remains the same.

5.4.2.2. SOCKETS

Lights Berry is listening to only one message from the backend for the light, that one is, *light-:id*. This contains a light in the message. After a security check to see if the *controller_id* of the light corresponds to the one stored in the database, it then updates the stored light and notifies Lights Duino to change the color of said light.

5.4.2.3. BLUETOOTH COMMUNICATION

Every time there is a change in a light from the app, the platform sends a buffer in a form of a characteristic with three values, the new RGB of the light.

Those buffers are as simple and small as they can be, that is why the server sends over just an array of integers between 0 and 255 to represent the color.

Finally, needs to be clarified that the intensity and the statuses vary the color number that is sent; an intensity of 0.5 creates a different color than an intensity of 1; also a status of off means that the number is 0. All the computation, is calculated in the backend or in the app and not in the micro-controller in order to minimize the size of the buffer and the computation of it making it faster and more reliable.

5.4.3. LIGHTS DUINO

This is the smallest components of all and consists of a small micro-controller. The one used to do this project is the *Arduino UNO*.

Lights Duino consists in:

- A snippet of code controlling the Bluetooth shield that manages the RGB LED.

The electronic part involving the light is pretty simple; it is an RGB LED with a Bluetooth shield that provides the BLE technologies required.

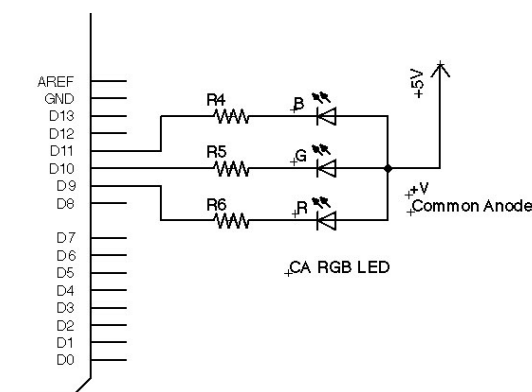


Figure 5. Arduino schematics of the connection of an RGB LED.

The RGB LED has a power supply of 5V with resistors around the pins in order to not burn the small sub LEDs that form the main light. The connection between the

shield and the pins is done, as seen in the figure 5, in the pins 9, 10 and 11. That means that in the micro-controller code, there is the possibility to use the functions *analogRead* or *analogWrite*, and not the digital ones. This is something needed to put a value in the LED between 0 and 1, challenges found using the digital architecture are shown in section 5 of the annex II.

5.4.3.1. BLUETOOTH IN THE ARDUINO

In order to use Bluetooth in the Arduino, there is a challenge. Looking at the specs^[2] of the *Arduino UNO*, there is no bluetooth available that can produce BLE. There has been two attempts in this project in order to accomplish Bluetooth; one with the old Bluetooth pattern (serial port connection) and one with the new BLE (peripheral connection) introduced in Bluetooth 4.0.

I. Serial port connection

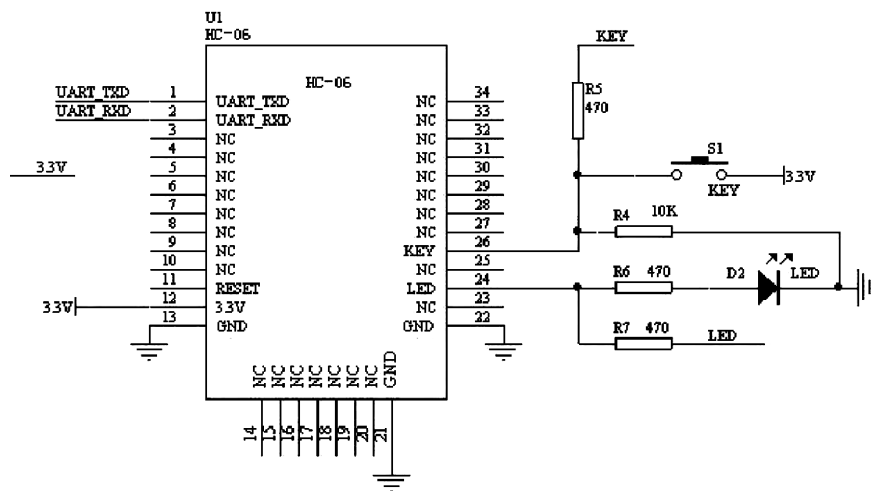


Figure 6. HC05 general module schematics.

In order to accomplish a simple Bluetooth connection, the *Arduino* uses in this case a HC05 serial module. Such module cannot handle peripheral connectivity, that means that the communication between both is unstable and cannot be guaranteed after a new connection or reconnection to send multiple messages with a padding of time.

Another challenge that appears is the need to build a script^[3] in shell in order to perform the pairing of the two devices for the first time, which, since the new *Bluez*^[4] version for *Linux*, such task is a bit more complicated having to perform operations into a dynamic system type. As a side note, the script solves that by putting delays and expects, awaiting for the terminal to finish its job in every task.

II. Peripheral connection

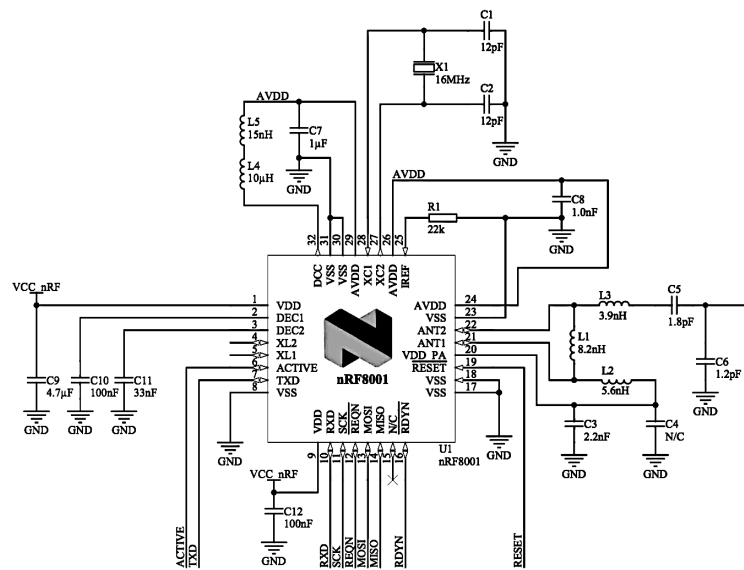


Figure 7. BLE Shield 2.1 main component, the nRF8001 BLE provider.

Such connection is accomplished by an external Bluetooth shield; the one used in this case is the BLE Shield 2.1^[5]. This component communicates with the Arduino through an ACI (Application Controller Interface), meaning that the pins perform the same operation as the shield, transforming the Arduino code into the a code the BLE shield can read with the provided libraries.

The peripheral connectivity allows multiple messages to be sent with different paddings of time sleeping the central module and the advertiser while they are not talking to each other. That is basically what Lights is and needs. Another fact that is important is the pairing of it, which is done automatically and within code, that

means that there is no need of external and internal Bluetooth libraries within the system and the kernel of *Linux*.

5.4.3.2. HANDLING OF THE LIGHT

The first thing the program of the Arduino does is to open the serial port; that makes the shield able to receive characteristics and connect to other devices. The serial port loops every second to try to receive new information. Once a characteristic is sent over a peripheral connection via Bluetooth the serial port checks the buffer sent from Lights Berry to see if it contains an array with three numbers from 0 to 255. The BLE shield reads those numbers with the function *ble_read()* provided by the creator's library and then it performs an *analogWrite* for those values to output in the LED.

5.4.4. LIGHTS APP

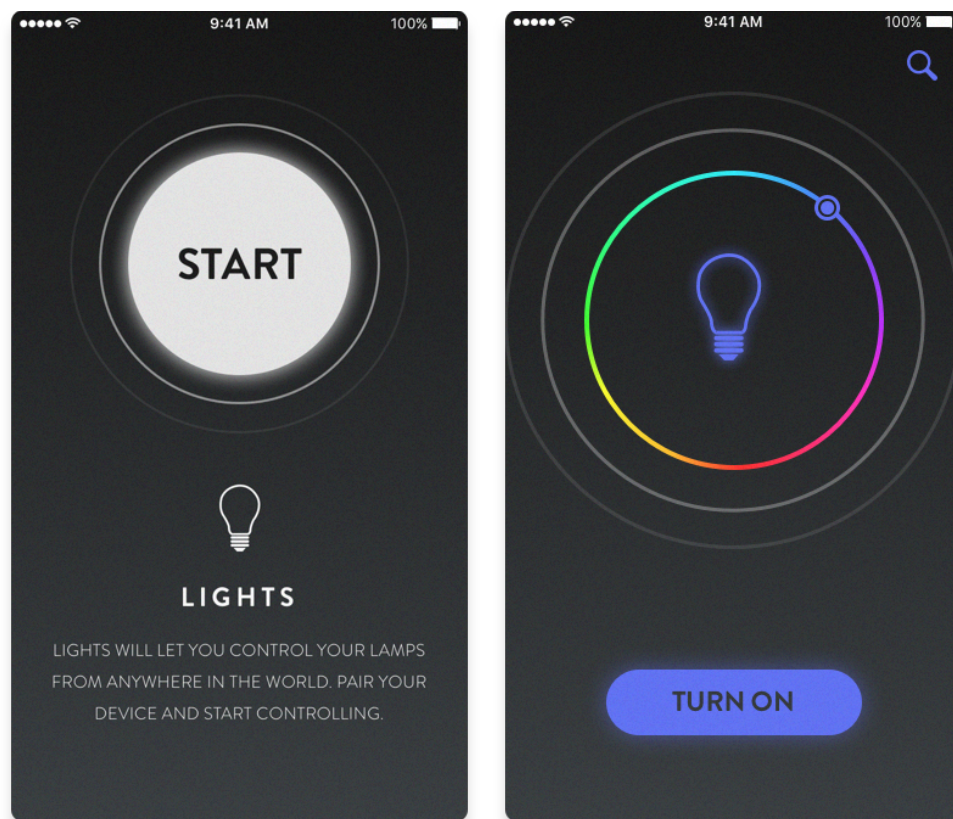
The app needs to communicate various things to the end user, easiness, cleanliness and being a straight to the point solution. This section explains the flow of the app and how the users experiences it by centering efforts into the UX and the implementation of it.

Lights contains:

- Internal database to store the information coming from the backend.
- Socket connector to emit and receive messages.
- Bluetooth hub to advertise the app even though it is also prepared to be a central module.
- Wheel of colors to select the desired color of the LED (all the color computation related remains within the app).

- Requester and networking layer to get information over HTTP from Lights Backend.

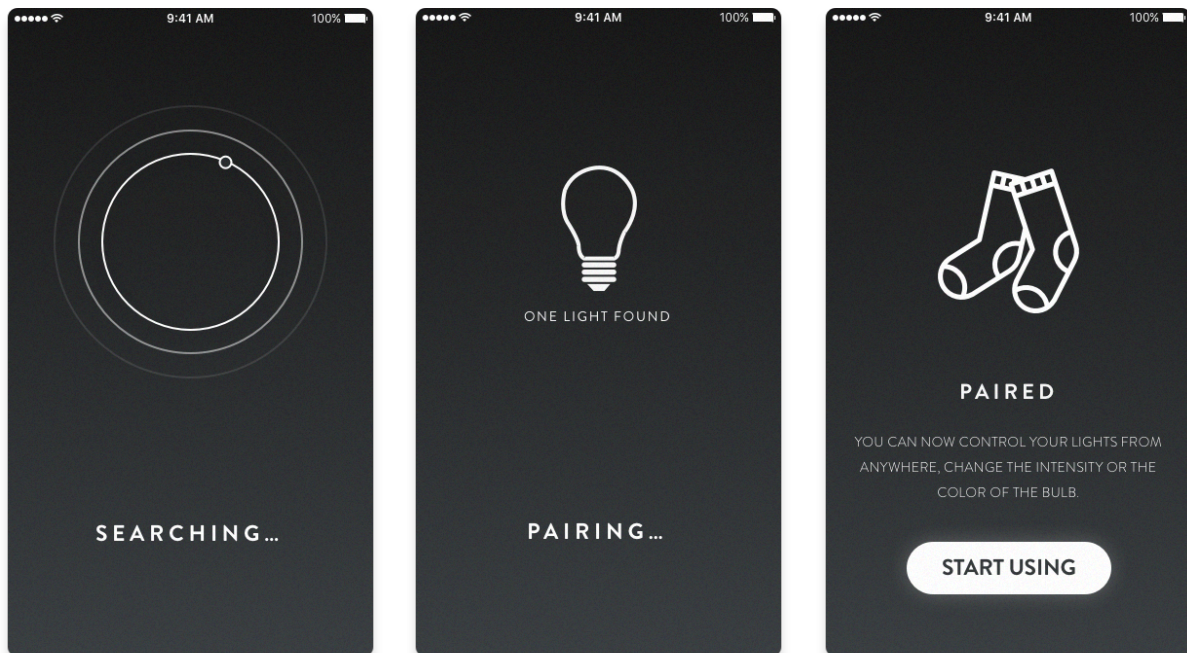
Before jumping straight to the flow, figures 8 and 9 show the two main screens of the app. Those are the most important screens since the one in the left is the one the user sees in the beginning when it opens the app for the first time and the one in the right is the one the user interacts with all the time. Those screens then need to show consistency and be easy enough for the end user to use.



Figures 8 & 9. The initial screen of lights (left) and the editing screen (right).

5.4.4.1. FLOW OF THE APP

The app presents other screens in addition to figures 8 and 9 in order to accomplish the two main functionalities of it; the pairing of the light via bluetooth as well as the fetching of it, and the controlling of it via sockets.



Figures 10, 11 & 12. (From left to right) Searching, pairing and paired screens of Lights.

I. First screen

After the user installs the app in the phone it is presented with the screen in figure 8. Such screen explains the purpose of the app and has a huge start button that lets the user start pairing the app with the platform. This screen has more importance than the others because is the first screen the user sees of the app. It needs to be functional and have a good design to engage the user with the product.

There is a ripple effect around the button hinting the action the user should take every time.

II. Searching screen

Once the user presses the start button, the app begins to look for peripherals around. As in iOS, there is the need to grant access to the Bluetooth of the phone in order to use it; the user is asked for it right before searching.

After that, the app simulates a radar to indicate that is performing the search of peripherals.

Is important to note that the phone is not trying to connect to anything. The phone is just advertising itself, it does not act as an advertiser; that basically means that the communication is a bit different, the phone does not send any message to the platform's Lights Berry server, the phone tells the server that it wants information and the server sends that information back within a characteristic.

III. Pairing screen

Once the connection is established, the server sends to the phone the token of the controller and the id of said controller. With that information, the app is able to start fetching the lights from the platform's Lights Backend's HTTP RESTful API, going to the next step when this one is ready.

As a side note, there is a designed error display where connection errors, internet problems and such are shown.

IV. Paired screen

Even though this is just another step that could seem unnecessary, in this screen the app downloads the content mentioned in point III and starts the connection to the backend via web sockets. When this is done, the end user is able to start controlling the lights after pressing the white button in the bottom of the figure 12.

V. Editing screen

The editing screen shall only do one thing and one thing only, editing the lights. Instead of doing some sliders to change the colors, the design opts to do something different, a bit more fun and easier to use. The wheel changes the colors when the user rotates it or changes the intensity when the user varies the radius of it.

With a big button to turn it on and off, the UI changes based on the color the user selects.

Finally, needs to be said that the user is able to search for new content or new lights tapping the button in the top right corner of figure 9; after that, the process starts again in point I.

5.4.4.2. INNER OPERATIONS OF THE APP

The app performs various inner operations under the hood. Some are convenience, like not showing the UI while there is no internet connection, some are UX related tasks like showing the pairing screen if there is no light saved yet in the persistency layer.

A part from those conveniences, the app downloads the content from the Lights Backend in the */lights* endpoint; when the response is valid, it saves the light into the persistency layer and connects via sockets with the library socket.io^[6]. Once this is done, the user is able to spin the wheel that performs the computation to get the position of the finger, building from that, the color for the LED light, sending a socket with the updated light object at the end.

As explained before, the app also advertises itself via Bluetooth to get the *token* and the *controller_id* in order for it to perform the Lights Backend requests.

5.4.5. INDUSTRIAL DESIGN

Lastly in the making of the lights, there are two pieces of industrial design that are done. A hub (the container of Lights Berry) and a light (the container of Lights Duino).



Figure 13. 3D render of the hub that controls the lights.

The plans from figure 13 are shown in section 1 of the annex III.

5.4.5.1. DESIGN OF THE HUB

For the design of the hub is necessary to do something simple, intuitive and easy to use. To do that the end user needs basically some key components in it:

- A reset button to restart the system in case there is a problem.
- An indicator that the device is ready to start pairing.

With that in mind, the hub also needs to contain a mini computer as the Raspberry Pi and have a connector to ethernet as well as to the current.

The design of it features a black beveled box with an indicator in the frontal part and the connections (ethernet and current) with the reset button in the back. In the top face there is a circular indicator for other events and future versions of the product.

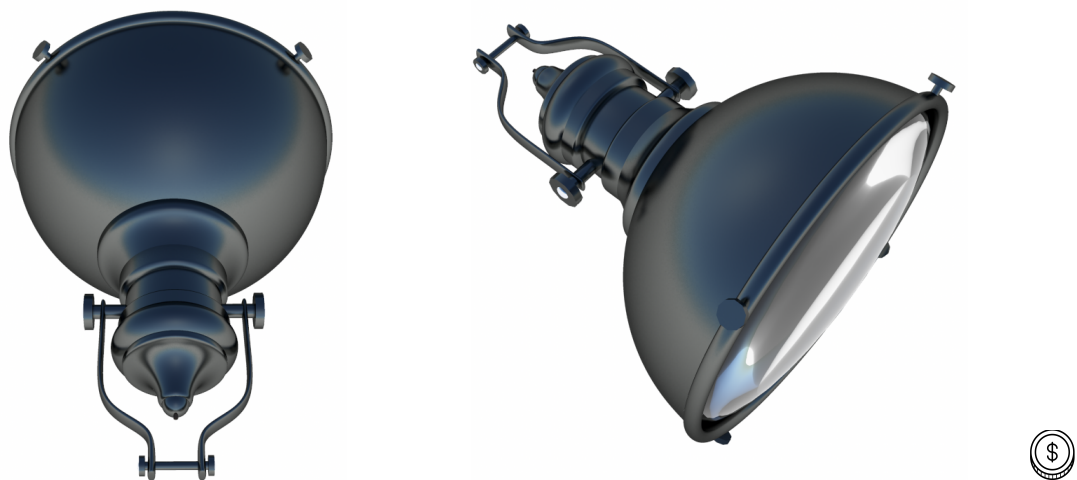
Such indicator could be used to show the current status of the lights, etc. Finally, the color of the hub is a black mat in the top and a plastic reluctant black in the sides.

5.4.5.2. DESIGN OF THE LIGHT

The design of the light has in mind that needs to be for the general public. The color chosen for the lamp is an industrial gray with a shape that reminds to an old light.

The lamp has multiple functions and positions and is designed with that intention in mind. It can be just a small lamp in the ground or in a table, featuring 4 screws in the face of it to stay in the desired position of the user, and it can also be used as a spotlight with the back part that subjects it.

With that introduction, the design of the lamp can be seen in the figures 14 and 15 with the plans of it appearing in the section 2 of the annex III.



Figures 14 & 15. Close up 3D renders of the light with a ground position. Size in comparison of a dollar coin.

The lamp is designed to have an E26 LED light type that is provided to have the hardware inside already available.

The light's materials are steel for the body of the lamp and stainless steel for the small screws that go in the middle of the big screws in the back. The front part of the light can be in crystal or plastic.

As a small detail and as seen in figure 16, the back of lamp has a small button that turns on and off the light so the user does not have to unplug it from the current if such user does not have the app around.

As seen in figures 14 and 15 with the comparison of the coin, the light is small enough because it needs to be portable and easy to hold for the user to be able to change the position of it. That is why the main body, from head to the feed is roughly 40 cm. Being the diameter of the eye 23 cm.

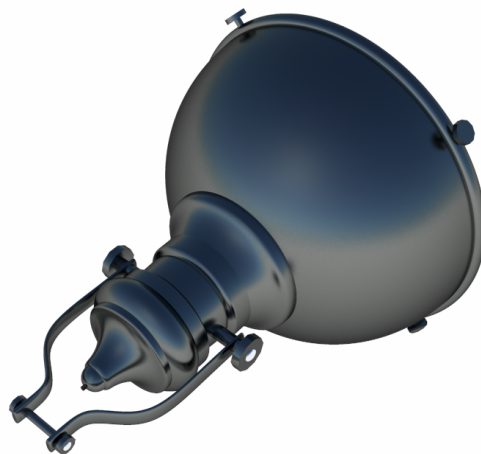


Figure 16. Detailed back of the lamp with the button to control it.

Aspects aside, the look of the light is industrial and trendy, with first class materials that makes it not for the geek market but for the general one instead. The ability of being a lamp separately without the hub adding the button in the back opens the end user and the market of it a little bit more, even though that is not the main intention of it.

5.5. REFERENCES

- [1]. GitHub. NodeJS startup script [2016, May 21]. Retrieved from: <https://gist.github.com/RamonGilabert/e15e91de0b5937e145bba4cca342c637>
- [2]. Arduino. Arduino UNO landing website [2016, May 21]. Retrieved from: <https://www.arduino.cc/en/main/arduinoBoardUno>
- [3]. GitHub. Bluetooth script to pair devices [2016, May 21]. Retrieved from: <https://gist.github.com/RamonGilabert/046727b302b4d9fb0055>
- [4]. BlueZ. BlueZ library release notes [2016, May 21]. Retrieved from: <http://www.bluez.org/category/release/>
- [5]. Red Bear. BLE shield landing page [2016, May 21]. Retrieved from: <http://redbearlab.com/bleshield/>
- [6]. Socket.IO. Library to send web-sockets from the web or from an app [2016, May 23]. Retrieved from: <http://socket.io>

6. TESTS AND RESULTS

There are numerous tests performed for the project to accomplish a maximum of 1% or less of error in all the listed cases, this introduction names a few.

The main tests across the project cover the most error prone situations, which for instance are:

- Connection of the app lost.
- Bluetooth connectivity lost.
- Device not found during search over Bluetooth.
- Multiple pairing of devices with the hub at once.

After those, it is important to cover the least error prone situations and the ones that are a bit harder to fix or are not fixable by the developer of the project since they relay in third party libraries and such.

- Web sockets fail in connection or in sending.
- The Bluetooth BLE 2.1 Shield stops working.

There are also stress tests in the emitting and broadcasting socket as well as the Bluetooth connection.

- Stress tests sending hundreds of requests per minute within a socket.
- Stress tests sending hundreds of requests per minute within the Bluetooth connectivity.

Finally, there are tests covering the whole process (when a new user connects for the first time the hub and the light), to see the rate of success in that scenario.

Each of the sub-sections has an extended explanation on why that particular test is needed and the success rate of it. Needs to be said that each test has a span of 8 to 16 tries for each situation, that, is added to all the debugging done during development, which makes it a final span of 8 to 60 tries for each.

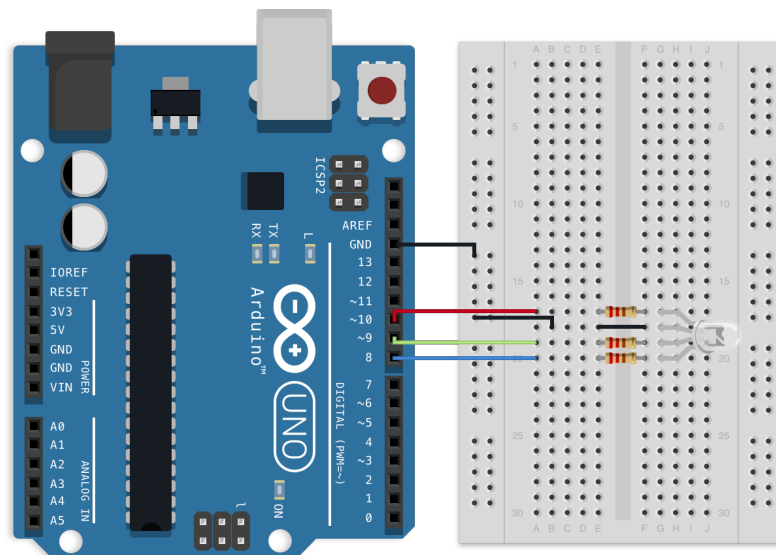


Figure 17. Arduino connection for the RGB LED.

All the tests are ran in debugging mode and with logs present at all times, that means that there is full control of the situations and if any problem occurs, it is tracked with different debugging tools.

6.1. ERROR PRONE SITUATIONS

In this case, an error prone situation represents anything that happens often or can happen for no apparent reason within the code, that could be the loss of internet connection, etc. such errors are separated into:

- Bluetooth connectivity problems such as disconnection, device not found, etc.
- Internet connectivity problems such as low internet, no internet at all, etc.

- Multiple devices trying to send sockets, connecting to the hub, etc. (stress tests) at once.
- Others, device not opening in a correct way, resetting not working, connection done in more than a period of time, etc.

6.1.1. BLUETOOTH TESTS AND RESULTS

Testing Bluetooth connectivity is hard in terms of accessibility; the connection could fail at any time even though the code is working; that means that there is no clear scenario to reproduce a bug or a problem. That makes the test be just difficult to perform and leads to the only type of test that makes sense, which is the error by trying.

First thing is to test if the Raspberry Pi connects to the phone via Bluetooth finding the light every time it resets.

I. Test over first connection - Light found.

The present test covers the cases where the hub is turned on and the light is either on or about to be.

Table 2. Pairing tests via bluetooth.

Time to find the light	Pairing success	Connection time after found
5s	✓	1s
6s	✓	1s
7s	✓	1s
5s	✓	1s
5s	✓	1s
6s	✓	1s
5s	✓	1s
6s	✓	1s

In order to do the test, the hub needs to be on (green LED with a value of 1). After that, a timer that tells the «Time to find the light» column starts to run. When the BLE shield turns on its LED, the test is done, it means that the connection is successful.

II. Test after first connection - Light controlled.

The present test has the intention to cover the case where the first test has passed and the user starts controlling the light with the phone.

Table 3. Pairing tests after the first connection is established.

Time to connect to the light	Fetch status from database	Pairing success
2s	✓	✓
1s	✓	✓
1s	✓	✓
2s	✓	✓
3s	✓	✓
2s	✓	✓
1s	✓	✓
1s	✓	✓

After the green LED of the hub is on, the user tries to turn on and off the light repeatedly checking when it behaves as it should, annotating the seconds elapsed in «Time to connect to the light» since such LED turned on.

The last column of this test annotates if the status of the light is fetched from the central database in the cloud.

III. Test over phone connection - Phone pairing for the first time.

The present test covers the pairing of the phone with the hub for the first time when the user downloads the app from the AppStore.

Table 4. Phone pairing rate of success for the first time.

Time to get the characteristic	Pairing success	Seconds to start controlling
3s	✓	1s
3s	✓	1s
4s	✓	1s
5s	✓	1s
2s	✓	1s
2s	✓	1s
3s	✓	1s
4s	✓	1s
4s	✓	1s
5s	✓	1s
2s	✓	1s

Since this is the first experience that the user has with the app and the platform, it is important for this test to be perfect. As a side note, the «Time to get the characteristic» is a bit bigger since the platform (for the first time ever), needs to find the light and connect to it too.

IV. Test over phone connection - Phone pairing for the X time.

The present test covers the phone connecting or trying to look for the hub again after the first connection has been successful; that is, when the user taps the search button in the upper right corner to search again for a new light.

Table 5. Phone pairing rate of success for the X time.

Time to get the characteristic	Pairing success	Seconds to start controlling
1s	✓	1s
2s	✓	1s
2s	✓	1s
1s	✓	1s
3s	✓	1s

Time to get the characteristic	Pairing success	Seconds to start controlling
2s	✓	1s
1s	✓	1s
4s	✓	1s

6.1.2. INTERNET TESTS AND RESULTS

The test in this section contains a poor yet inexistent internet connection from the device of output (iOS phone) to the Lights Backend in order to control the light, showing the seconds of duration based on the MB/s of downloading and uploading that the phone has.

Table 6. Low or inexistent internet connection tests.

Duration in seconds to perform	KB/s	Success in changing
3s	500	✓
4s	500	✓
5s	500	✓
3s	500	✓
5s	500	✓
6s	500	✓
7s	500	✓
4s	500	✓

The profile used to perform this test is the one in the following table.

Table 7. Parameters of the internet connection used in the tests.

	Downlink	Uplink
Bandwidth	1 MBPS	500 KBPS
Packets dropped	10%	10%
Delay	500ms	500ms

6.1.3. STRESS TESTS

There are two ways to perform the stress tests; the first one is by performing lots of requests to the server in the cloud, the other one is stressing the Raspberry Pi, thus the light with multiple connections at once.

The following tests present the number of connections per minute that are done in order for the test to pass.

I. Stress to the cloud

To perform such operation, the backend reduces resources (cores, power, etc.) and sleeps more often; that makes each request to be slower since it has to either wait for the backend to be awake or to finish processing its current requests. In order to fix possible errors, there is going to be an increase in the number of cores and power of the server having it always awake depending on how many users Lights has; the price for this aspect is shown in section 7.2.3 in the column of costs derived from hosting.

Table 8. Stress connection tests to the cloud.

Requests per minute	Time of response	Connection success
60	1s	✓
120	1s	✓
180	1s	✓
240	1.5s	✓
300	1.5s	✓
360	1.5s	✓
420	2s	✓
480	2s	✓
540	2s	✓
600	2s	✓ (98% of the times).

The connection success rate is given by whether the table in the database changes when the socket is performed and if the backend does not crash or sends back a 503 error (Internal Server Error) or a socket not connected error.

As seen in the last requests, 10 connections per second to the backend are too much causing the server to crash. By adding more resources (which is configured to do automatically), this gets fixed.

II. Stress to the light

The present test is related to the previous one because after there has been a success in the database, Lights Backend tries to connect to Lights Berry. Such connection to the hub is what this test measures. The result of it is given by the change of the light. It can be either a positive result (the light changes or turns state) or a negative one (the light does not change status or it shuts down because of overload).

Table 9. Stress connection tests to the hub.

Requests per minute	Time of response	Connection success
60	1.5s	✓
120	1.5s	✓
180	1.5s	✓
240	2s	✓
300	2.5s	✓
360	2s	✓
420	2.5s	✓
480	2s	✓
540	2.5s	✓
600	2.5s	✓

The last requests have a 100% success rate relative to the success rate of the one in Lights Backend with the same number of requests.

6.2. PROCESS TEST

It is so important to make a great product in the exterior of it as it is to make a perfect development and execution in the interior. The process test measures the first time ever the user connects the hub for the first time and downloads the app from the store altogether.

The way this test measures its results is by measuring time and success rate of the different phases. A first timer that tells when the hub has posted itself (the controller), a second timer that tells if the light is connected, letting the third timer tell if the phone got a successful connection. After that, the success rate is represented by if the light changes state from a phone's command.

Table 10. First time process success rate.

Hub ready	Light found	Phone found	Success
5s	1s	1s	✓
6s	2s	1s	✓
3s	2s	1s	✓
4s	1s	2s	✓
5s	2s	2s	✓
4s	2s	2s	✓
3s	1s	3s	✓
4s	2s	1s	✓
3s	1s	1s	✓
4s	2s	3s	✓

The tests are performed in a good internet connection environment because the hub is connected to the ethernet. The first process «Hub ready» is the most complicated one, thus it takes more time, that is because the app needs to initialize, check databases, fetch information from Lights Backend, etc. other than that, the BLE shield does a really good job finding and performing Bluetooth connections.

7. BUDGET AND ENGINEERING

The present point, and explicitly section 7.1, explains the cost of the technologies and the design of a light that is prototyped and built before the design phase, using section 7.2 to show the costs derived from porting such prototype into mass production. Such sections are shown depending on if they are all time costs depending on production or one time costs depending on development.

The industries before the prototype include:

- Creative and concept design.
- Development and engineering to have a working object.
- Design phase with materials and different tests.

The industries after the prototype include:

- Fabrication of different sub-prototypes with different end results and the same design idea to check, after user testing, the best and most accepted end result.
- Industry for the quality of the service in a stress test of the points stated in section 6.1.3 as well as a stress test for the materials and the construction of it.
- Perfectionism and iterating engineering.
- Global manufacturing.

This section tries to simplify the budget of the project understanding the construction of Lights into mass production and comparing mass prices over different provides over the world and facing them towards the estimates.

Needs to be said that all the work that is done before and after the prototyping phase is a one man work and hour, so this budget does not show how a cross functional team of any sort would perform doing the same operations.

7.1. GENERAL OVERVIEW OF LIGHTS

Before jumping into the estimations and the actual cost in mass production, this section explains the man hours involved and the components that are being used in Lights. Thus, this section explains the before industries, which simplify the after ones.

7.1.1. CREATIVE AND ARCHITECTURAL PHASE

In every creation of a platform there is a lot of creative phase in order to decide what the product needs to end up being, which concepts are involved to it and even the architecture of each of the most valuable ones.

The hours spent on this section are hours that normally could not be billed or added to the product if this was bound to be sold to a client or to a pitch of ideas; however, since this is has an end result to a product with a general public, the hours are counted as billable.

Following the Gantt diagram shown in table 1 in section 4.3; the design phase, creative phase, and so on, take place over the course of 3.5 months, with the discussions for the architecture taking place all over the project. The first estimation for this phase is at around **60 hours** to have all the architectural system prototyped in paper before starting; those 60 hours include a small buffer to discuss further solutions and solve different problems derived from scalability, etc. while developing.

7.1.2. DEVELOPMENT, TESTING AND ENGINEERING PHASE

While building a product there are hours of research, prototyping, internal design and actual building and making of the product. This subsection explains in detail each of the hours of the engineering process divided into the actual sub products of the platform.

The hours spent to build the prototype, test and engineer are **220 hours** in total and are divided into different user stories within each epic, each epic represents one topic, which represents a big theme or a big mark in the platform. Such user stories follow the standard agile pattern^[1] used as a standard in the development industry to deliver in a faster and lighter way.

I. Lights Backend (70h)

As an admin I want to have a database to see all the controllers and products. (17.5h)

As an admin I want to have a socket central module in where to manage connections. (15h)

As an admin I want to have a basic RESTful API with the controllers and products. (27.5h)

As a user I want the backend to be configured within the terms of lights. (10h)

II. Lights Berry (55h)

As an admin I want to have the control to be able to get and send sockets. (15h)

As an admin I want to be able to have a Bluetooth central module ready to get, receive and send information via such communication. (30h)

As an admin I want to have a requester pattern to communicate to a RESTful API. (8h)

As a user I want to have a way to know that the product is ready to use. (2h)

III. Lights Duino (5h)

As an admin I want to be able to parse a buffer of 3 integers and represent them in an RGB LED. (3h)

As a user I want to have a light and the electronics to see the results from the app. (2h)

IV. Lights app (60h)

As a user I want a fast way to edit and control the color and status of the light. (20h)

As a user I want to have the ability to pair to a device to get information. (15h)

As a user I want to connect via sockets to the platform. (5h)

As a user I want to be able to request and save in the persistency layer a light. (15h)

As a user I want to be notified if there is no internet connection. (5h)

V. Scripts (15h)

As a user I want to have Lights Berry in connection to the current. (5h)

As an admin I want to have a way to interact with BluetoothCTL. (10h)

VI. Testing (15h)

As a user I want to have a crash free platform and product. (15h)

7.1.3. DESIGN, PROTOTYPE, TINKERING

The design phase contains the initial wireframes and final concept for the app as well as the 3D prototypes and renders for the hub and lamp; all of that is presented in sections 5.4.4 and 5.4.5. The estimation in hours also includes the learning of 3D design and development done in Cheetah 3D^[2].

The general estimation for the design phase, prototyping and tinkering of it while the development process is on course is of around **40 hours** in total. 15 hours for the app and 25 for the actual 3D renders, learning, plans (shown in annex III), etc.

7.2. TOTAL COST ESTIMATIONS

After seeing the actual hours divided into each phase, the estimations of the costs is presented in this sub-section containing both the before prototyping and the after one taking different manufacturing prices from different providers as an example.

7.2.1. ONE TIME COSTS

All the prices shown in table 11 display the general price to develop the platform and the product, thus, this is a one time cost and not something that is applied to every light or every end product separately.

Note that this price does not include maintenance in the product as well as further development; such task is done if enough lights and products are made and there is enough money to continue with the development of it.

Table 11. Estimated cost of the industries involved before the prototype phase is completed.

	Hours	Price (1 hour)	Total price
Creative	60	30	\$1,800.00
Engineering	220	40	\$8,800.00
Design	40	35	\$1,400.00
General testing	20	20	\$400.00
			\$12,400.00

All the prices an hour are shown as the standard in the industry for a starter project.

7.2.2. PER LIGHT COST

With that, there is the need to add the prices of the different components used in the project, such as the BLE shield, the Raspberry Pi as a micro computer, the Arduino as the micro-controller, etc.

Table 12. Cost of the distinct components per light.

	Quantity	Price
BLE Bluetooth Shield	1	\$19.90
Arduino UNO	1	\$29.95
RGB LED Bulb	1	\$10.00
General components	1	\$2.00
		\$61.85

Even though the light and the hub are treated as one in production and manufacturing, its inner components are different from one another, that creates two tables.

Table 13. Cost of the distinct components per hub.

	Quantity	Price
Raspberry Pi	1	\$35.00
General components	1	\$2.00
		\$37.00

Such prices show the cost of just one of the lights and not the price into mass production which is cheaper than individually.

Finally, there is the need to add the prices for the materials used to build the lights as well as the general electronic components.

Table 14. Estimated cost of the materials used to produce each light.

	Quantity	Price
Steel	400 gr	\$0.86
Plastic	50 gr	\$0.02
Cables	1 m	\$1.00
General components	-	\$2.00
		\$3.88

General components include screws, trims, etc. The other prices are shown from major wholesale industries in different places around the world making a mean cost and price out of it^{[3][4][5]}.

Table 15. Estimated cost of the materials used to produce each hub.

	Quantity	Price
Plastic	300 gr	\$0.12
Cables	2 m	\$2.00
General components	1	\$1.50
		\$3.62

7.2.3. MASS COST

Finally, there is the need to calculate costs of some of the parts of the platform based on the mass production that it has, that is, cost of the hosting depending on how many users^[6], cost of the packaging and general manufacturing, etc.

Table 16. Estimation of the cost in mass production of the different values.

Packs	Manufacturing	Hosting (mo)	Before industry	Maintenance	Cost per light
1	\$280.00	\$0.00	\$12400.00	\$0.00	\$3,170.00
10	\$2,800.00	\$7.00	\$12400.00	\$0.00	\$380.18
100	\$26,000.00	\$7.00	\$12400.00	\$500.00	\$97.27
1,000	\$50,000.00	\$100.00	\$12400.00	\$10,000.00	\$18.13
10,000	\$500,000.00	\$500.00	\$12400.00	\$50,000.00	\$14.07
100,000	\$5,000,000.00	\$750.00	\$12400.00	\$100,000.00	\$12.78
1,000,000	\$50,000,000.00	\$1500.00	\$12400.00	\$250,000.00	\$12.57
10,000,000	\$500,000,000.00	\$5000.00	\$12400.00	\$650,000.00	\$12.52

There is an special consideration in the table which is that one pack is 3 lights and 1 hub and all of them have the same price so the total cost is divided by 4 to simplify terminology.

The final manufacturing costs (that include packaging) are an estimation based on different providers called in Norway.

Table number 16 shows that the price of a light and thus, a hub tends to 12.5\$ in a real mass production environment. That represents more people working on maintenance, making the company evolve, new technologies in the product, improvements in the platform, etc.

Such prices do not include the general marketing of it, making it a bit hard to estimate since the starting point is a crowdsourcing campaign where the marketing is done by itself. Transport costs derived from the selling of products from the campaigns are included as a plus in the price and do not affect the cost of the product; that also applies to taxes in different countries derived from the light, being the taxes from the manufacturing process included as per Norway (25%) and is susceptible to change based on the country in which such production is performed.

7.3. MARKET STUDY AND GENERAL PUBLIC

Is difficult to predict the market that a light like this one is bound to have. Depending on how much stores, marketing, etc. it might have more or less sells. However, the idea for this project is to make a crowdsourcing campaign out of it, which reports documented cases of other successful campaigns ran and can give a rough estimate of the product income, inversion to make, etc.

It is seen in section 7.2.3 that the price that a light should target to is of around 14\$ or less; that means having at least 30000 lights and 10000 hubs, which is the equivalent to 10000 packs on sell.

This would make the total cost, if the intention is to have lights at the target price mentioned, of around **\$600,000** ($10000\$ \cdot (14\$ \cdot 4) + 40000\$$); such price includes a bit of a margin for the business but not the 30% that is intended to be added.

After a quick look in the most popular crowdsourcing campaigns, it is seen that such projects normally pledge something around \$400,000 to \$600,000^{[7][8]}. Such lights cost

around 19 - 20 dollars each, which makes it more expensive than the price lights is targeting to (without transport and marketing costs since it is a given that such crowdsourcing campaigns provide it already and the transport is added as a plus on top).

A part from that though, needs to be said that the intention of the lamp is not just to be smart but also to be a normal use case light; that is since it has buttons to interact to within the hardware; that potentially opens up the public of it to a wider range and, when categorizing the product, it could be recognized under design and not only under technology.

7.4. FINAL COST ESTIMATION

Table 16 shows that, with a production tending to more than 4000 units of the product, Lights starts to be an affordable object and starts competing with the other products in the market seen in section 3.

Making the product tend to more units and imagining an scenario where the market is strong enough, a pack of Lights would cost 50\$ without the business margin, adding a margin of around 30%, different marketing campaigns for it in different websites, etc. and costs derived from other aspects, the final price of it would be at around 80\$ a pack, meaning 3 lights and one hub.

That makes the product affordable enough and with an amortization due to the 10000th pack.

7.5. REFERENCES

[1]. MGS. Agile user stories, how to write them [2016, May 28]. Retrieved from: <https://www.mountaingoaftware.com/agile/user-stories>

[2]. Cheetah3D. 3D software for design [2016, May 28]. Retrieved from: <http://www.cheetah3d.com>

[3]. Alibaba. Stainless steel wholesale [2016, May 29]. Retrieved from: http://www.alibaba.com/product-detail/carbon-steel-or-stainless-steel-wire_60364005977.html?spm=a2700.7724838.0.0.z0qiqW

[4]. Alibaba. Polycarbonate plastic wholesale [2016, May 29]. Retrieved from: http://www.alibaba.com/product-detail/UV-Reflective-Wholesale-Makrolon-Solid-Polycarbonate_60401155754.html?spm=a2700.7724838.0.0.v7XEZd&s=p

[5]. Alibaba. Cables wholesale [2016, May 29]. Retrieved from: http://www.alibaba.com/product-detail/Europe-Plug-RF-03_60406625917.html?spm=a2700.7724838.0.0.F4GzKZ

[6]. Heroku. Heroku pricing [2016, May 29]. Retrieved from: <https://www.heroku.com/pricing>

[7]. Indiegogo. Smart bulb campaign in a crowdsourcing website [2016, May 29]. Retrieved from: [https://www.indiegogo.com/projects/world-s-most-affordable-wi-fi-smart-bulb# /](https://www.indiegogo.com/projects/world-s-most-affordable-wi-fi-smart-bulb#/)

[8]. Kickstarter. Smart lamp campaign in a crowdsourcing website [2016, May 29]. Retrieved from: <https://www.kickstarter.com/projects/2128753402/fluxo-the-worlds-first-truly-smart-lamp>

8. MOTIVATION AND CONCLUSIONS

8.1. MOTIVATION

The idea of the project is born from the passion to solve a problem that, as engineers we are encountering more and more nowadays, diversity and specialization. My knowledge is based in an electronic engineering degree combined with software development and, in those two fields, the diversity of ideas is growing every day more and more. From an electronic perspective there are more and more optimized versions of new components every day, cheaper hardware, more memory or more capacity means more innovation; as per software, there are more and more programming languages aiming to become the next big thing, frameworks to do new components, etc. but there is quite nothing that unifies what we have learnt in the past 50 years from software development with the knowledge that we are getting nowadays from hardware and electronics.

This project aims to unify my knowledge to create something that has smartness, union, and components working together to create something that just 10 years ago was so costly and difficult that we could not think of.

8.2. WHY LIGHTS

Lights has two main focuses. First, to proof that the future is merging different fields within engineering such as computer science and electronics, but most importantly, Lights is born to proof that home automation, connected devices and the internet of things can be cheap and easy enough for people to use at their homes targeting a general public and not only the geek one.

The first question that raises is then, why a smart lamp? Lights, as mentioned in this report has two servers that talk to each other; if something abstract enough from that

paradigm can be extracted easily and something else can be imported into such platform, anything can be smart and part of the home automation deal. A thermostat could be connected into the current architecture changing some of the basic parameters and then the project would be a bit different although the idea would be the same. That means that with such paradigm, this project aims to create a platform rather than just an end project; that led to just one question, which is the device to be connected, the answer is a light for its visual feedback and easiness in the use and installation.

8.3. CONCLUSIONS

It is hard to know where to stop when building something as generic and straight forward into the future as this. After the whole process explained in this document, Lights turned to be a generic open source platform with an inducted product that can control lights from anywhere in the world, but the fact that anybody can edit it and customize it to make it their own, makes this something that cannot be imagined or described in just a small report.

From the first wheel invented until this project; lots of years of innovation and inventions have put technology in our day to day lives; more and more people interact with their devices now, 9 years ago we could not imagine holding a super computer in our hands while now, we wear it in our wrists. Every day there is a new couple of words together forming a new one, AI, VR, IoT, etc.

Breaking technologies disrupt the market being the next big thing until a new technology appears and, within this world where we live, fast delivered solutions that solve real problems in a light and easy way is something engineers, designers and companies around the world are and will try to aim for.

Within this context, Lights creates something the world is going towards to, specialization while diversity within industries. In 2016 the future tells engineers that they have to be able to jump into a new technology within weeks if they want to keep their jobs, be in the last trends or even be able to carry on designing or engineering products. This level of change is what makes Lights truly important; the fact that it has so many industries together, so many technologies involved; software, hardware, electronics, design, development, engineering, planning, economics, financing and many more and just to name a few.

After more than 350 hours developing, researching, creating, writing and designing, Lights is what it was intended to be, a prototype that can be used from anywhere in the world, connected with all the easiness available, with a good design aiming to go for the general public and ready to be sent into a crowdsourcing campaign; thus, it completes the goals that were set in the beginning of this project back in October.

There are lots of new ideas coming every day that can be added into a project like this, new technologies, languages or frameworks that improve the communications, new shields or super tiny computers that give velocity and easiness in the implementation or even new standards to push and add new users. With this, it is a given that Lights is on and does not stop here with this spec; this is just the MVP of something that can truly be the next big thing within the Internet of Things; that field within software and electronics that is just about to get started.

9. GENERAL REFERENCE LIST

9.1. BIBLIOGRAPHY

Gordon McComb [2013]. Arduino Robot Bonanza: McGraw-Hill.

Giancarlo Fortino, Paolo Trunfio [2014]. Internet of Things Based on Smart Objects: Springer.

Guillermo Rauch [2012]. Smashing Node.js: JavaScript Everywhere: Smashing.

Mike Cantelon, Marc Harter, TJ Holowaychuk, Nathan R [2014]. Node.js in action: Manning.

9.2. WEBLIOGRAPHY

ABI research. 1.5 Million Home Automation Systems Installed in the US This Year [2016, May 15]. Retrieved from: <https://www.abiresearch.com/press/15-million-home-automation-systems-installed-in-th/>

Apple. Developer documentation for CoreBluetooth [2016, May 25]. Retrieved from: https://developer.apple.com/library/ios/documentation/NetworkingInternetWeb/Conceptual/CoreBluetooth_concepts/CoreBluetoothOverview/CoreBluetoothOverview.html

Alibaba. Stainless steel wholesale [2016, May 29]. Retrieved from: http://www.alibaba.com/product-detail/carbon-steel-or-stainless-steel-wire_60364005977.html?spm=a2700.7724838.0.0.z0qiqW

Alibaba. Polycarbonate plastic wholesale [2016, May 29]. Retrieved from: http://www.alibaba.com/product-detail/UV-Reflective-Wholesale-Makrolon-Solid-Polycarbonate_60401155754.html?spm=a2700.7724838.0.0.v7XEZd&s=p

Alibaba. Cables wholesale [2016, May 29]. Retrieved from: http://www.alibaba.com/product-detail/Europe-Plug-RF-03_60406625917.html?spm=a2700.7724838.0.0.F4GzKZ

Arduino. Arduino UNO landing website [2016, May 21]. Retrieved from: <https://www.arduino.cc/en/main/arduinoBoardUno>

BlueZ. BlueZ library release notes [2016, May 21]. Retrieved from: <http://www.bluez.org/category/release/>

Cheetah3D. 3D software for design [2016, May 28]. Retrieved from: <http://www.cheetah3d.com>

Gartner. 6.4 Billion Connected "Things" Will Be in Use in 2016, Up 30 Percent From 2015 [2016, May 17]. Retrieved from: <http://www.gartner.com/newsroom/id/3165317>

GitHub. Bluetooth script to pair devices [2016, May 21]. Retrieved from: <https://gist.github.com/RamonGilabert/046727b302b4d9fb0055>

GitHub. Bluetooth struct within Lights [2016, May 21]. Retrieved from: <https://github.com/RamonGilabert/Lights/blob/master/Lights/Lights/Library/Bluetooth/Bluetooth.swift>

GitHub. Lights Backend by Ramon Gilabert [2016, May 16]. Retrieved from: <https://github.com/RamonGilabert/Lights-Backend>

GitHub. Lights Berry by Ramon Gilabert [2016, May 16]. Retrieved from: <https://github.com/RamonGilabert/Lights-Berry>

GitHub. Lights by Ramon Gilabert [2016, May 16]. Retrieved from: <https://github.com/RamonGilabert/Lights>

GitHub. Lights Duino by Ramon Gilabert [2016, May 16]. Retrieved from: <https://github.com/RamonGilabert/Lights-Duino>

GitHub. Lights Editing view [2016, May 21]. Retrieved from: <https://github.com/RamonGilabert/Lights/blob/master/Lights/Lights/Editing/Views/EditingView.swift>

GitHub. Noble library [2016, May 21]. Retrieved from: <https://github.com/sandeepmistry/noble>

GitHub. NodeJS startup script [2016, May 21]. Retrieved from: <https://gist.github.com/RamonGilabert/e15e91de0b5937e145bba4cca342c637>

Heroku. Heroku pricing [2016, May 29]. Retrieved from: <https://www.heroku.com/pricing>

Home (Apple). An app to interact with the internet of things devices [2016, June 19]. Retrieved from: <http://www.apple.com/newsroom/2016/06/apple-previews-ios-10-biggest-ios-release-ever.html>

HomeKit (Apple). A platform for the Internet of Things [2016, May 17]. Retrieved from: <http://www.apple.com/ios/homekit/>

Honeywell. Honeywell history and inventions [2016, May 15]. Retrieved from: <http://twincities.honeywell.com/honeywell-history-and-minnesota-heritage/>

Hue (Philips). A smart lamp and bulb connected with an intermediary hub [2016, May 17]. Retrieved from: <http://www2.meethue.com/nn-no/>

Indiegogo. Smart bulb campaign in a crowdsourcing website [2016, May 29]. Retrieved from: [https://www.indiegogo.com/projects/world-s-most-affordable-wi-fi-smart-bulb# /](https://www.indiegogo.com/projects/world-s-most-affordable-wi-fi-smart-bulb#/)

Instructables. NodeJS and web-sockets tutorial [2016, May 20]. Retrieved from: <http://www.instructables.com/id/Easy-NodeJS-WebSockets-LED-Controller-for-Raspberr/>

Kickstarter. Smart lamp campaign in a crowdsourcing website [2016, May 29]. Retrieved from: <https://www.kickstarter.com/projects/2128753402/fluxo-the-worlds-first-truly-smart-lamp>

Link Labs. Comparison between BLE and Bluetooth [2016, May 25]. Retrieved from: <http://www.link-labs.com/bluetooth-vs-bluetooth-low-energy/>

MGS. Agile user stories, how to write them [2016, May 28]. Retrieved from: <https://www.mountaingoatsoftware.com/agile/user-stories>

Nest (Alphabet). A smart thermostat [2016, May 17]. Retrieved from: <https://nest.com>

Red Bear. BLE shield landing page [2016, May 21]. Retrieved from: <http://redbearlab.com/bleshield/>

Servicelab. Control an Arduino with NodeJS over Bluetooth [2016, May 20]. Retrieved from: <http://servicelab.org/2012/12/12/wirelessly-control-your-arduino-with-nodejs-over-bluetooth/>

Socket.IO. Library to send web-sockets from the web or from an app [2016, May 23]. Retrieved from: <http://socket.io>

UC3M. How to write a bibliography or a reference list [2016, May 31]. Retrieved from: http://portal.uc3m.es/portal/page/portal/biblioteca/aprende_usar/como_citar_bibliografia#sitios

UC3M. How to write a scientific document [2016, May 14]. Retrieved from: http://docubib.uc3m.es/CURSOS/Documentos_cientificos/Normas%20y%20directrices/UNE_50135=ISO%205966.pdf

UC3M. How to write a scientific document [2016, May 14]. Retrieved from: http://docubib.uc3m.es/CURSOS/Documentos_cientificos/Normas%20y%20directrices/UNE_50135=ISO%205966.pdf

UNC. Formatting guidelines in a thesis [2016, May 14]. Retrieved from: <http://gradschool.unc.edu/academics/thesis-diss/guide/format.html>

University of Lleida. Thesis law from the University of Lleida [2016, May 14]. Retrieved from: http://www.eps.udl.cat/docs/info_acad/normatives/tfg/actual/normativa/Reglament_TFG_i_TFM.html

Wikipedia. Retina display explanation [2016, May 22]. Retrieved from: https://en.wikipedia.org/wiki/Retina_Display

ANNEX I

INDEX

1. Color wheel (iOS).....	1
2. Advertiser Bluetooth connectivity (iOS).....	3
3. Central module Bluetooth (NodeJS).....	4
4. Database in Lights Berry (PostgreSQL)	5
5. Socket connectivity (NodeJS)	6
6. References.....	7

1. COLOR WHEEL (IOS)

The following code shows the main components to build a wheel like the one in the Lights App.

Code 1-AI. Color wheel coded for an iOS app.

```
func createWheel() -> CGImageRef? {
    let dimension: CGFloat = Dimensions.size
    let bufferLength: Int = Int(dimension * dimension * 4)
    let bitmapData: CFMutableDataRef = CFDataCreateMutable(nil, 0)

    CFDataSetLength(bitmapData, CFIndex(bufferLength))

    let bitmap = CFDataGetMutableBytePtr(bitmapData)
    let final = Int(Dimensions.size)

    for x in 0 ..< final {
        for y in 0 ..< final {
            let point = CGPoint(x: CGFloat(x), y: CGFloat(y))
            var hsv: HSV = (hue: 0, saturation: 0, brightness: 0, alpha: 0)
            var rgb: RGB = (red: 0, green: 0, blue: 0, alpha: 0)
            let color = saturation(point)
            let saturate = color.saturation

            if saturate < 1 {
                let alpha: CGFloat = saturate > 0.992 ? (1 - saturate) * 100 : 1

                hsv = (hue: color.hue, saturation: saturate, brightness: 1, alpha: alpha)
                rgb = convertHSV(hsv)
            }

            let offset = Int(4 * (point.x + point.y * Dimensions.size))

            bitmap[offset] = UInt8(rgb.red * 255)
            bitmap[offset + 1] = UInt8(rgb.green * 255)
            bitmap[offset + 2] = UInt8(rgb.blue * 255)
            bitmap[offset + 3] = UInt8(rgb.alpha * 255)
        }
    }

    let colorSpace = CGColorSpaceCreateDeviceRGB()
    let dataProvider = CGDataProviderCreateWithCFData(bitmapData)
    let bitmapInfo = CGBitmapInfo(rawValue: CGBitmapInfo.ByteOrderDefault.rawValue
        | CGImageAlphaInfo.Last.rawValue)
    let reference = CGImageCreate(final, final, 8, 32, final * 4,
        colorSpace, bitmapInfo, dataProvider, nil,
        false, .RenderingIntentDefault)

    return reference
}
```

Other helper functions are used to convert values from the HSV color space to the RGB color space as it is seen in section 1 of the annex II. The important part, though,

is to explain how to build a color wheel within milliseconds having to go within so many pixels as a Retina display^[1] has.

The main important part of the piece of code is the fact of interacting directly with the C compiler, creating an async buffer with the *CFMutableDataRef*. With that and two for loops to represent both the X and the Y axis that go through all the points of it until the desired size of the wheel, the program calculates the exact value in the color spectrum with the *saturation* function and adds it to the first array of mutable data that we have previously created.

After such data is created, it performs an operation with the *CGImageCreate* global function to create a small image of one pixel by one pixel of the given color calculated, as said, with the relative position within the wheel.

Finally, needs to be said that the helper functions used and also the gestures of the wheel can be seen in the footnote link that shows the file containing the called *EditingView*^[2].

2. ADVERTISER BLUETOOTH CONNECTIVITY (IOS)

To create a Bluetooth connectivity within an iOS phone, the app asks for the permission to get to the Bluetooth hardware and, after that and importing *CoreBluetooth*, the developer can start being both a central module or an advertiser.

Code 2-AI. Generic bluetooth advertiser for an iOS app.

```
let peripheralManager = CBPeripheralManager(delegate: self, queue: queue)
peripheralManager.scanForPeripheralsWithServices([CBUUID(string: Constants.service)],
options: nil)

func peripheral(peripheral: CBPeripheral, didDiscoverServices error: NSError?) {
    guard let services = peripheral.services else { return }

    for service in services {
        peripheral.discoverCharacteristics([CBUUID(string: Constants.characteristic)],
            forService: service)
    }
}

func peripheral(peripheral: CBPeripheral, didDiscoverCharacteristicsForService service:
    CBService, error: NSError?) {
    guard let characteristics = service.characteristics else { return }

    for characteristic in characteristics {
        peripheral.setNotifyValue(true, forCharacteristic: characteristic)
    }
}

func peripheral(peripheral: CBPeripheral, didUpdateValueForCharacteristic characteristic:
    CBCharacteristic, error: NSError?) {
    if let data = characteristic.value, string = String(data: data, encoding:
        NSUTF8StringEncoding) {
        // Do what is needed with the string received.
    }
}
```

As in the previous section there are a bunch of helper functions that perform the non essential operations^[3].

In this snippet of code, it is shown the basic *delegate* methods called when discovering peripherals around as an advertiser, from the service, to the characteristic with the message within it. In this case, this message contains the controller and the token to be used to fetch information from Lights Backend.

3. CENTRAL MODULE BLUETOOTH (NODEJS)

To create a Bluetooth as a central module in NodeJS, there is the need to import a library called Noble^[4]. After a connection, Noble is able to start scanning.

Code 3-AI. Bluetooth central module in NodeJS.

```
noble.on('stateChange', function(state) {
  if (state === "poweredOn") {
    noble.startScanning();
  }
});

noble.on('discover', function(peripheral) {
  if (peripheral.advertisement.localName === "BLE Shield") {
    var address = peripheral.address;
    connect(peripheral, "713d0003503e4c75ba943148f18d941e")
    .then(function(characteristic) {
      // With the characteristic, save the address of the Arduino and create a light.
    } else if (peripheral.advertisement.localName === "Lights") {
      connect(peripheral, "7DAB97504510410CB030D5597D3EBE6D".toLowerCase())
      .then(function(characteristic) {
        var buffer = new Buffer(
          controller.token.toString() + ' ' + controller.id.toString(), "utf-8");
        characteristic.write(buffer, false);
      });
    }
  });
});
```

When something is discovered, weather if it is a BLE Shield (the micro-controller component) or Lights (the iOS app), it does one operation or another^[5].

Needs to be said that Noble sends buffers in order to be able to transmit the information; an example of that is when it finds an app; the token of the controller and the id of it conform the characteristic that is sent back.

4. DATABASE IN LIGHTS BERRY (POSTGRESQL)

The snippet of code 4-AI, creates the database structure for Lights Berry. It has the two entities needed to do so, *Controller* and *Lights*.

Code 4-AI. PostgreSQL snippet that creates the Lights Berry database.

```
CREATE TABLE Controller(  
  id INT NOT NULL,  
  phone_id INT NOT NULL,  
  created DATE NOT NULL,  
  updated DATE NOT NULL,  
  address STRING,  
  PRIMARY KEY (id)  
);  
  
CREATE TABLE Lights(  
  id INT NOT NULL,  
  status BOOL NOT NULL,  
  intensity FLOAT NOT NULL,  
  red FLOAT NOT NULL,  
  green FLOAT NOT NULL,  
  blue FLOAT NOT NULL,  
  created DATE NOT NULL,  
  updated DATE NOT NULL,  
  token STRING NOT NULL,  
  address STRING,  
  controller_id INT NOT NULL,  
  PRIMARY KEY (id)  
);  
  
ALTER TABLE Lights ADD CONSTRAINT lights_fkey FOREIGN KEY (controller_id) REFERENCES  
Controller(id)
```

It is interesting to point out the last line of it, where, adding a constraint with a foreign key, such code adds a relation to the Lights table from the Controller one in order for each Light to have just one parent and the Controller to have multiple lights instead.

5. SOCKET CONNECTIVITY (NODEJS)

In order to accomplish socket connectivity in NodeJS there are lots of libraries, to choose one there is the need to look into which one performs best in iOS; after some research the chosen library is socket.io^[6].

Code 5-AI. Socket receiver in Lights Berry (NodeJS).

```
var socket = io.connect('https://lights-backend.herokuapp.com', { reconnect: true });

socket.on('connect', function() {
  console.log('A light connected to the central server.');
```

```
});

socket.on('light-' + controllerID, function(light) {
  berry.light(light.light);
});

socket.on('new-ios-light-' + controllerID, function(light) {
  if (!controlled) {
    controlled = true;
  }
});
```

The connection via sockets is pretty simple with such library. There is the need to connect first and, after that, the receiver can get messages^[7] whereas the emitter can send them. It is seen in the code and implementation of Lights Backend that such connection broadcasts messages to sockets instead of creating a 1 on 1 communication with them, that is just in case there was multiple users with the app on controlling one light at the same time for instance. The example code above shows a receiver socket.

6. REFERENCES

- [1]. Wikipedia. Retina display explanation [2016, May 22]. Retrieved from: https://en.wikipedia.org/wiki/Retina_Display

- [2]. GitHub. Lights Editing view [2016, May 21]. Retrieved from: <https://github.com/RamonGilabert/Lights/blob/master/Lights/Lights/Editing/Views/EditingView.swift>

- [3]. GitHub. Bluetooth struct within Lights [2016, May 21]. Retrieved from: <https://github.com/RamonGilabert/Lights/blob/master/Lights/Lights/Library/Bluetooth/Bluetooth.swift>

- [4]. GitHub. Noble library [2016, May 21]. Retrieved from: <https://github.com/sandeepmistry/noble>

- [5]. GitHub. Lights Berry's Bluetooth class [2016, May 23]. Retrieved from: <https://github.com/RamonGilabert/Lights-Berry/blob/master/app/classes/bluetooth.js>

- [6]. Socket.IO. Library to send web-sockets from the web or from an app [2016, May 23]. Retrieved from: <http://socket.io>

- [7]. GitHub. Lights Berry's Sockets class [2016, May 23]. Retrieved from: <https://github.com/RamonGilabert/Lights-Berry/blob/master/app/classes/socket.js>

ANNEX II

INDEX

1. General concepts	1
I. What is a RESTful API?	1
II. What is an endpoint in an API?	2
III. Details of an HTTP method.	2
IV. Web sockets vs HTTP.	3
V. What is and why Heroku?	3
VI. What is and why Postgres?	4
VII. Bluetooth, advertising, peer to peer.	4
VIII. Theory of colors.	6
IX. What is a persistency layer?	8
2. Other components.	9
3. The making of in numbers.	10
4. Testing the project.	11
5. Challenges	12
6. References.	13

1. GENERAL CONCEPTS

This section explains general concepts used across the explanations in technological detail that are useful to note. Some are generic terms in backend development, some generic terms in the technologic world.

I. What is a RESTful API?

To answer this question, is best to start by defining what an API is; API stands for Application Programming Interface and, while there are lots of types of APIs, this explanation is centered in just a web based type system. An API represents a set of protocols or tools that make building software easy and accessible for other clients to interact with it.

As it is seen during the explanation of the making of, this project uses a shield to use BLE technologies in the Arduino. The shield creator has facilitated an API for the developers to access such shield, it has some methods, properties, etc. that the developer can use.

In the case of the project, the API is the one that is provided by the server, in this case the Lights Backend to the iOS app and to Lights Berry in order for those last two to communicate with the central backend.

However, the question then is, what does the backend facilitate to the, in this case, clients? Basically a RESTful pattern for them to be able to fetch, edit, add or delete items in the cloud's database.

A RESTful API provides the clients with a set of endpoints (routes, explained later) in where to *POST*, *PUT*, *PATCH*, *DELETE* or *GET* items from the database. This means that there is a separation of concerns between where the main data is, which is in the backend, and where it is represented. And that is a backend with an API.

II. What is an endpoint in an API?

An API endpoint defines a contract for the communication between a client and a backend, this means that, the backend, since the API lives there, proposes a contract with a set of routes (for instance */lights*) to the clients, in those routes, the API does something based on the HTTP method that the request is going with, this can be deleting an entity in the database, etc. In the previous point it is said that those methods can be *POST*, *PUT*, *PATCH*, *DELETE* or *GET*.

In an API that supports fetching */lights* a *GET /lights* call would return normally a JSON response with an array of light objects.

An endpoint can have multiple paths inside its own route, that means that */lights* can have an id to identify a specific light. That is represented like this: */lights/:id*. In this case, doing a *GET /lights/:id* would return just the light with the given *id* in the request.

III. Details of an HTTP method.

HTTP is an application protocol that creates the basic communication over the web. In the endpoint section, it has been said that when somebody requests something the API responds with something else, can be JSON, can be HTML, can be anything.

In order for the API to return something there needs to be a transmission protocol, in this case TCP (Transmission Control Protocol) to a particular port; that means that when the server starts it opens a port somewhere in the cloud in this case. When the client requests something to that port the server is able to return something.

The methods that form the HTTP pattern have been already said in the two previous points above.

Finally, needs to be said that an HTTP method has three main parts to do a request, the method as explained before, some headers and the body. The headers are general objects that should go to every request but that does not make sense to put into the body. The body is the request in general if needed. A header to request or post JSON could be the key *Content-Type* with the parameter *application/json*; the server then knows based on the object in the header that the return type for the response should be JSON.

IV. Web sockets vs HTTP.

While most of the internet is using HTTP, web sockets (referred as sockets in most of this document) are a relatively new pattern that make internet communication between devices and clients really fast and easy even though it breaks the pattern of the safe communication over the net. Instead of having to create a connection every time, a socket establishes a connection to another socket once; in this case one server works as a socket and a client works as another one. Once this handshake is done, both sockets can start sending messages to one another. This last part of the sentence is crucial because in the HTTP requests the server is not prepared to talk back to the client to request some information. With sockets, the server can listen to any message the client can send.

V. What is and why *Heroku*?

Heroku is the chosen application platform to host the server^[1]. Said platform works as a server that contains multiple applications inside. *Heroku's* difference over some of the most famous competitors such as AWS^[2] by Amazon is the deploying system; once something is merged into the master branch in *GitHub* in this case, *Heroku* deploys a new version of the server and makes it available right away, no need to copy and paste code, etc.

VI. What is and why Postgres?

PostgreSQL is the database language used to develop the project. Even though there is no particular reason on why Postgres, it has worked really well during the whole project with great support in Linux.

In order to check the status of the database, entities, tables, etc. A software developed by an indie developer has been used, the name of the application is *Postico*^[3].

VII. Bluetooth, advertising, peer to peer.

Bluetooth is one of the essential components of Lights and also one of the most complicated topics to cover. This section explains the two types of Bluetooth connectivity that exist and the one chosen in Lights.

Without entering into too much detail of the history of Bluetooth, this explanation talks about the differences between a pairing system in Bluetooth and a BLE (central module) system where other devices act as a peripherals.

In order to explain how Bluetooth works and used to work, this explanation uses as an example a heart rate measurement system.

In this case, such system would be able to pair the device with the phone and that phone would remember that that device was paired via Bluetooth. The next time that heart rate measurement system would appear next to the Bluetooth module, the phone would remember that the address of the device is something it should be paired to, so it would try to connect to it, having always a bridge between those two devices while in range.

While this is fine, Bluetooth 4.0^[4] brought BLE, or Bluetooth Low Energy. This kind of communication, while significantly similar to the old Bluetooth has the difference

in that the central module, in this case the phone would sleep the connection while not being in use. That would make the battery life significantly better.

The problem with BLE is that not every component supports it since it is a relatively new thing, components such as the Arduino UNO do not have support for it nor cheap serial modules of Bluetooth. That makes things a bit more complicated and that is why third party libraries and components help in this case.

The way BLE works now is with peripherals, services and characteristics. Basically there is a peripheral which is the device and another peripheral which is the phone. One of them acts like an advertiser while the other acts like the central module. The difference between those is basically that the advertiser looks for central modules to connect whereas the central module waits for advertisers to appear in a search and in range.

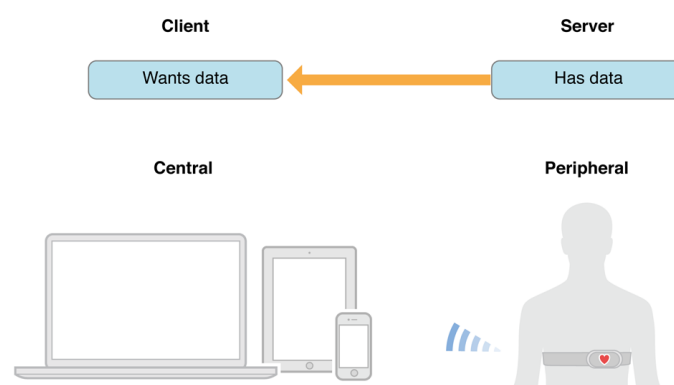


Figure 1-AII. Example of a generic connection for a health solution using BLE.

Following with the example of the heart system, the phone starts to look for an advertiser peripheral around; once the heart system starts to advertise itself, the connection is established and then a service connects to the central module (phone). A service is a part of a peripheral that contains characteristics^[5]. As an example, the heart measurement system has a service called *Heart rate service*, such service contains multiple characteristics with different messages inside, those could be the

body sensor location, the heart rate measurement, etc. In the Lights' case, this can be for instance the light status, the light intensity, the light color and so on.

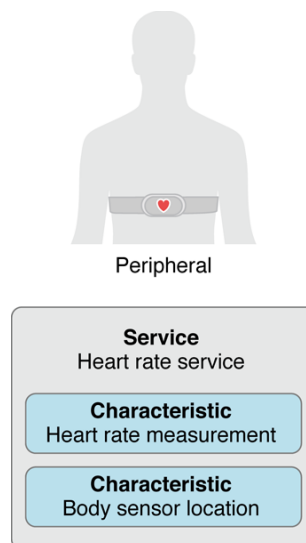


Figure 2-AII. Characteristic examples within a peripheral connection.

Those characteristics and the messages inside are the buffers that the central module, or the advertiser receive.

Another key difference between the normal Bluetooth and BLE is the quantity of data that it is intend to transport. While Bluetooth consumes a lot of battery, it also transports a lot of data between devices. That is a problem that BLE does not have since characteristics are intended to conduct small amounts of data, the heart rate in a given time, the light color, etc.

VIII. Theory of colors

The theory of colors is something that needed to be used within the project when building the app, as shown in the making of the product, the app has a wheel of colors; those colors are built based on the location in the screen of the wheel, size, radius, etc.

Such wheel also needs to represent colors that can be ported to a LED. That fact is why is necessary to understand how a color is built, different types of colors, etc.

There are multiple types of ways to represent a color, the most natural ones are the RGB, standing for Red, Green and Blue, the CMYK model, standing for Cyan, Magenta Yellow and Key (black) or for instance the HSB, standing for Hue, Saturation and Brightness.

Each combination of those colors or parameters can build any color in the spectrum. This explanation does not enter into much into the CMYK model, used a lot for printed models. Instead, it explains the RGB model and how to port that model into the HSB model, used to represent the wheel in Lights.

An RGB color^[6] is the one that is created from the colors it stands for, as said before, red, green and blue. Those colors from a range between 0 and 255 can create any color. Additionally, there is another scale called RGBA, which also includes the alpha of the color. This is not used to measure the intensity of such color in this case or the opacity in a normal case. Example of colors are for instance the white color *RGB 255 255 255* or the black, *RGB 0 0 0*.

In the other hand, an HSB color is represented by the hue, saturation and brightness of a color. One by one, hue is the property of a color that represents its hue, meaning, its general value, there are 6 hues in the spectrum: red, orange, yellow, green, blue and violet; those are the pure bright colors. Saturation means the intensity of a color, how intense a color is in its powerfulness, in photography, more saturation means more powerful colors. Finally the brightness of a color means what it stands for, how much brightness that color gives. While black gives zero brightness, white gives all the possible brightness.

In order to build a wheel, it is easier to do so with an HSB spectrum, that is because the representation of the colors are done with HSB, where the saturation goes with

the radius and the hue goes with the shape of the circle. Combining those values in an algorithm, this one is able to create a wheel of colors.

But the question still remains, how to convert that HSB color that the wheel needs to be built to a RGB color that the LED can represent?

Code 1-AII. HSL (HSB) to RGB conversion for iOS.

```
var second = B < 0.5 ? B * (1 + S) : (B + S) - (S * B)
var first = 2 * B - second

var R = 255 * converter(first, second, H + (1 / 3))
var G = 255 * converter(first, second, H)
var B = 255 * converter(first, second, H - (1 / 3))

func converter(first, second, hue) {
    if (hue < 0) hue += 1
    if (hue > 1) hue -= 1
    if ((6 * hue) < 1) return (first + (second - first) * 6 * hue)
    if ((2 * hue) < 1) return second
    if ((3 * hue) < 2) return (first + (second - first) * ((2 / 3) - hue) * 6)

    return first
}
```

If the saturation is 0, the color is white because the radius is 0, the same happens when the brightness is 1. There is a formula that represents all the other cases.

IX. What is a persistency layer?

The persistency layer in an application is represented by the database inside the app. In iOS, normally done in Core Data^[7] and nowadays with Realm^[8] among others.

For this project, the persistency layer is built upon the *NSUserDefaults* that Apple has in iOS. *NSUserDefaults* are a much more simple way to persist something if that something is simple enough, one object, one primitive value, etc.

As what needs to be persisted in this app is just an array of lights, the defaults is the best and easiest place to save it.

2. OTHER COMPONENTS

Is worth mentioning some of the components that have been useful to develop while doing the platform and that work as a separate library or component for other people to use. As stated already all the code and design for Lights is open source and available in GitHub^[9].

The first component is Socket^[10], a client to send web-sockets in the format of socket.io to any connection. Editing some parameters and putting the code lets you send a message. This is useful before having the app to test if the sockets are working or not without having to build a special client for it.

The second component is Ripple^[11], a convenience library to create ripples in a UI's app, really simple made and easy to use.

The third component is Bluetooth^[12], this library makes the task of configuring a Bluetooth device within your app really easy, weather you are advertising or looking for peers as a central module, trying to receive messages or anything similar to it. Bluetooth creates the central managers for you.

3. THE MAKING OF IN NUMBERS

After 2 servers, an iOS app, some shell scripts for Linux, some libraries, Arduino code, and many more, the prototype is up and running. Some of the numbers that led to finish the project are the following.

- The project has more than **15000 lines** of code written by the author and more than 20000 lines combined with external libraries.
- The project has **7 different programming languages** (NodeJS, PostgreSQL, Swift, Shell, HTML, CSS and C).
- The whole project including design is **open source** and available in GitHub.
- The project includes electronics and software going together as is the case of the Bluetooth connectivity.
- The project has created **3 different libraries** available after its creation, listed in 5.9.
- People from over **7 different countries** have helped in the concept, idea and making of the project.
- The production of the code and the design has been done during more than **4 months** every afternoon.

4. TESTING THE PROJECT

This section explains how the process of testing the platform is, since there are lots of components involved, there are also lots of ways to test each of them individually and in group. As section 6 explains, there are different things to test in a platform of this magnitude.

For the Lights Backend, as it is just a straight forward API, the local version of it is just running in the computer and, when working, deploying to *Heroku*. In order to test such API there is a RESTful client named *Paw*^[13] that has been used. To use this client the user just enters some parameters about the HTTP request and makes the call towards the API, the app returns the response from the backend.

For Lights Berry the situation is different. Postgres does not behave in the same way in Mac OS X than it does in Linux which makes it a bit harder to test since the configuration needs to be different. The way it is solved is to do the simple backend in Mac OS X pushing a working version to GitHub. After that, over *SSH* to the local IP of the Raspberry Pi just connect to it, clone the git repository, run the builder to install all the dependencies and run the server over *SSH*. That way the interaction is done easily, testing the code by putting logs across the server.

To test the sockets, the component mentioned in section 2 of the annex II is used, the one called Socket; that makes the process easier because there is no need to build a client or an app in order to see some results before jumping into the actual production code.

The native app is tested multiple times doing the process of changing lights, etc. over and over to check if it works 100% of the times. The app has also been tested while developing, which makes it less error prone than the other solutions.

5. CHALLENGES

Before the server had Bluetooth communication the light was controlled by the Raspberry Pi itself, the problem with that is basically that the Raspberry Pi does not have multiple analog pins. it works with digital pins, with an output of either 1 or 0. In order to accomplish different colors there is the need to give values like RGB 0.5 0.1 0.7 and not only 1 or 0. The way to solve this is to make pulses of different frequencies to achieve the color that the RGB value represents. It is what is known as PWM modules.

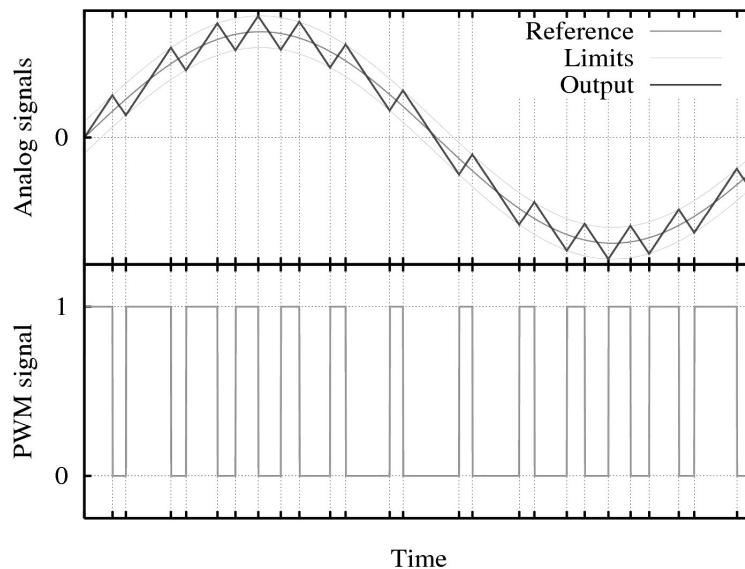


Figure 3-AIII. Graph comparing a PWM signal with an analog output.

This is obviously solved by just letting the Arduino connect the light over its analog ports. Those can produce different outputs than either 0 or 1.

6. REFERENCES

- [1]. Heroku. About page of Heroku [2016, May 24]. Retrieved from: <https://www.heroku.com/about>
- [2]. Amazon. Amazon Web Services [2016, May 24]. Retrieved from: <https://aws.amazon.com>
- [3]. Egger Apps. Postico app explanation [2016, May 25]. Retrieved from: <https://eggerapps.at/postico/>
- [4]. Link Labs. Comparison between BLE and Bluetooth [2016, May 25]. Retrieved from: <http://www.link-labs.com/bluetooth-vs-bluetooth-low-energy/>
- [5]. Apple. Developer documentation for CoreBluetooth [2016, May 25]. Retrieved from: https://developer.apple.com/library/ios/documentation/NetworkingInternetWeb/Conceptual/CoreBluetooth_concepts/CoreBluetoothOverview/CoreBluetoothOverview.html
- [6]. RGB. Information about the RGB pattern [2016, May 26]. Retrieved from: <http://rgb.to>
- [7]. Apple. Developer documentation for CoreData [2016, May 27]. Retrieved from: <https://developer.apple.com/library/watchos/documentation/Cocoa/Conceptual/CoreData/index.html>
- [8]. Realm. Documentation and explanation of what Realm is [2016, May 28]. Retrieved from: <https://realm.io>
- [9]. GitHub. GitHub page of the author [2016, May 29]. Retrieved from: <https://github.com/RamonGilabert>

[10]. GitHub. Socket repository [2016, May 29]. Retrieved from: <https://github.com/RamonGilabert/Socket>

[11]. GitHub. Ripple repository [2016, May 29]. Retrieved from: <https://github.com/RamonGilabert/Ripple>

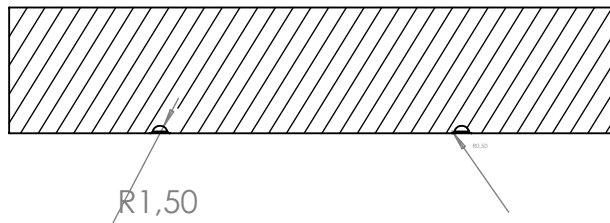
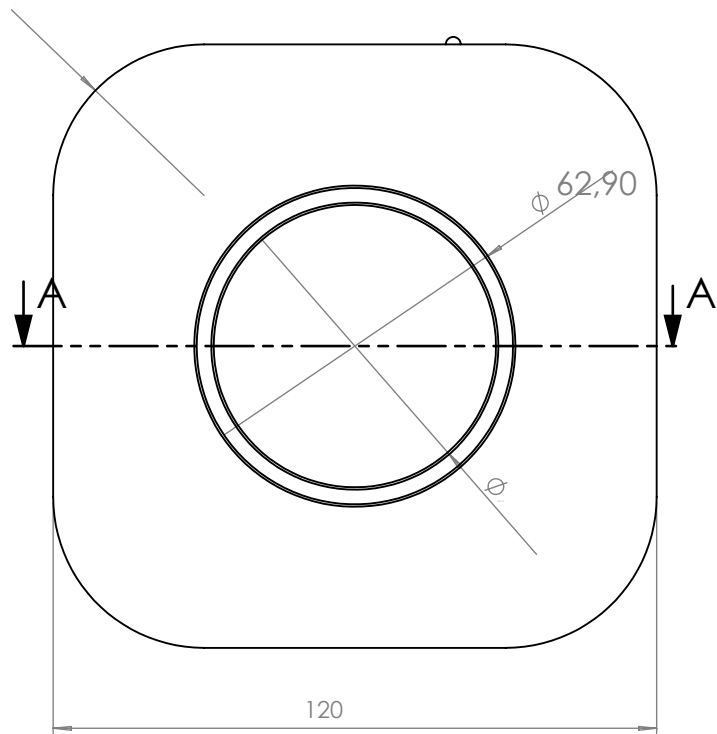
[12]. GitHub. Bluetooth repository [2016, May 29]. Retrieved from: <https://github.com/RamonGilabert/Bluetooth>

[13]. Lucky Marmot. Explanation of what Paw is [2016, May 29]. Retrieved from: <https://luckymarmot.com/paw>

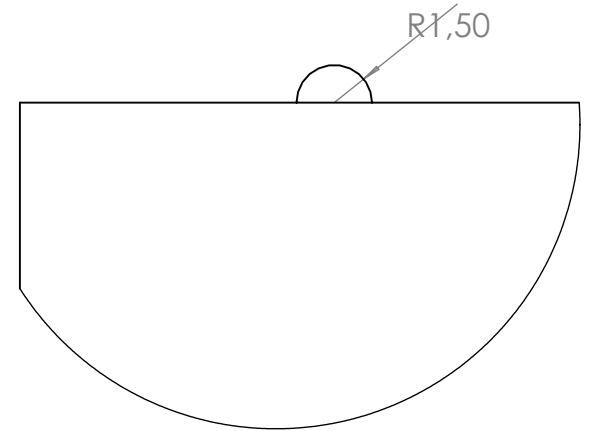
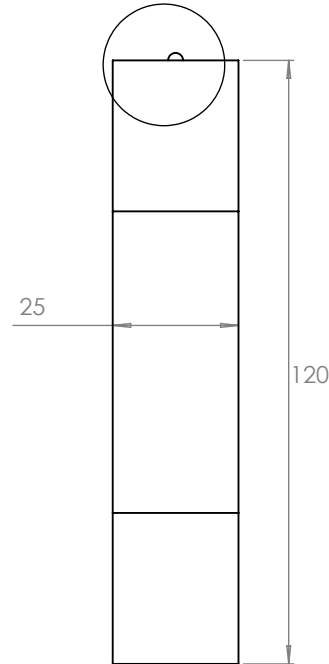
ANNEX III

INDEX




1. The hub plans.....	1
2. The lights plans.....	2

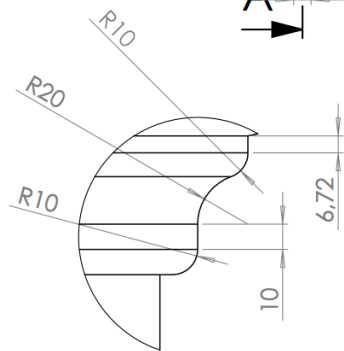
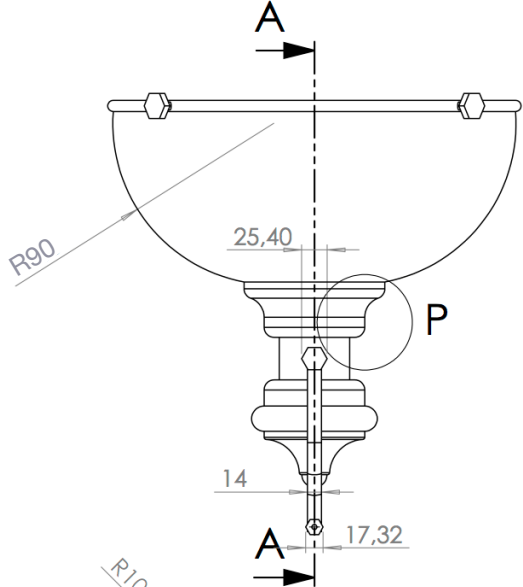
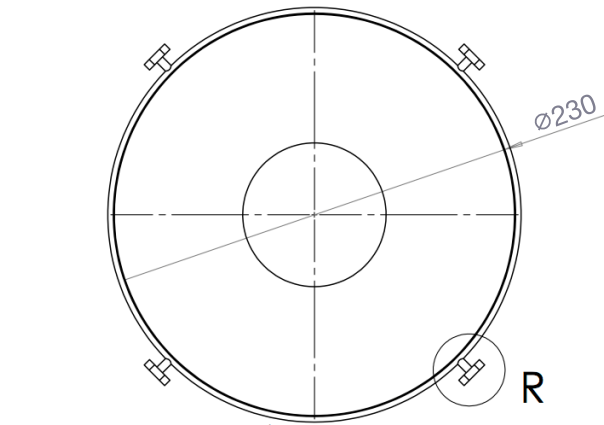


SECTION A-A
SCALE 1 : 1

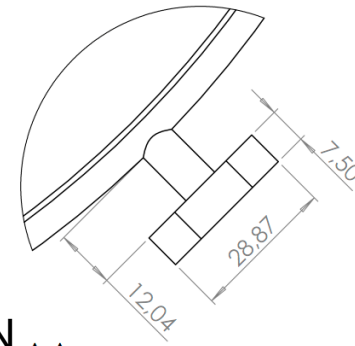


DETAIL B
SCALE 5 : 1

Material:	General tolerances (mm)		Not indicated  0,5 x 45°	 Universitat de Lleida Escola Politècnica Superior
Mass:				
Thermic treatment:	Dimension	Tolerance		Description:
Superficial treatment:	0 - 10 10 - 50 50 - 200 > 200	± 0,1 ± 0,2 ± 0,8 ± 1		
Projected:	Not indicated rugosity:			Reference:
Ramon Gilabert Llop		Scale: 1:1.5 Format: A3		
If not indicated the opposite measurement are shown in mm			Hub	
			Revision: 1	Sheet: 1/5

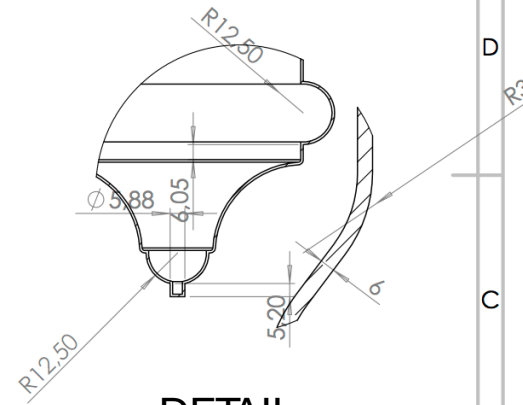
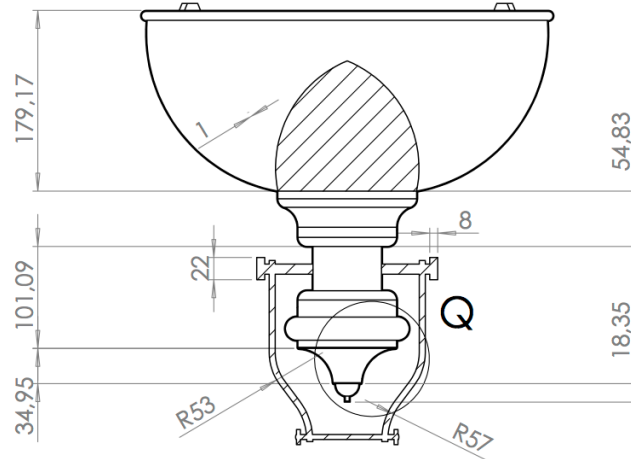


DETAIL P
SCALE 1 : 2



DETAIL R
SCALE 1 : 1

SECTION A-A
SCALE 1 : 5



DETAIL Q
SCALE 1 : 2

Material:	
Mass:	
Thermic treatment:	
Superficial treatment:	
Projected:	
Ramon Gilabert Llop	
If not indicated the opposite measurament are shown in mm	

General tolerances (mm)

Not indicated

R 0,2

0,5 x 45°



Description:

Reference:

Lights

Revision: 1

Sheet: 1/5