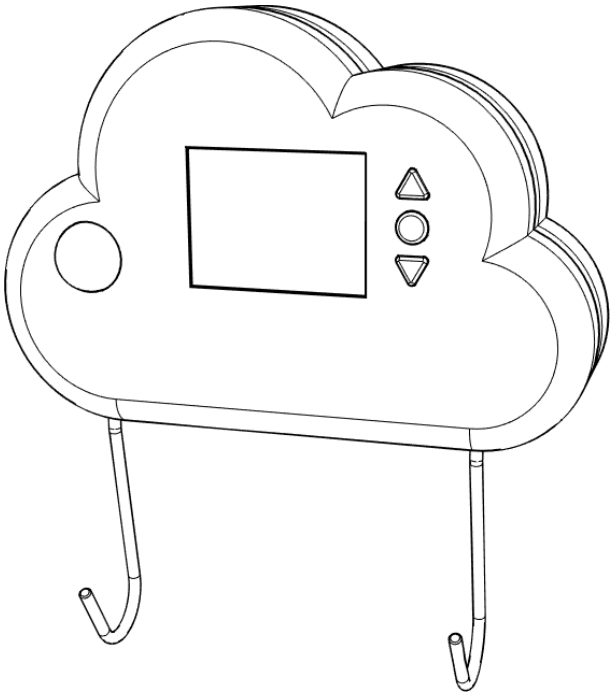
Cloudy Report

Physical Computing 2016

Giacomo Del Rio, Giuseppe Mendola
June 2016



Contents

1 Introduction

The 2016 course of Physical Computing at USI was aimed to teach us the basics of microcontroller programming (with an Arduino board), 3D printing and Android development: as a final project, we were asked to build a real prototype of a device that can exploit the power of these technologies.

Our first idea was about a device that can remind people of taking their umbrella when going out in the morning: a device capable of gathering weather forecast for the upcoming day and use them to emit an alarm when the user was next to the door in the morning.

This idea was further refined by shaping the device as a real umbrella stand, capable of holding two umbrellas and sense their presence. Then the concept of "remind about" was generalized to include arbitrary reminders, not only about umbrellas, in arbitrary moments of the day. Finally we tought about how to add security and configurability to the system and we decided to build a server to hold user information and preferences that are uniquely identified by the UUID of a bluetooth device associated with him. The name chosen for our device is *Cloudy*, in honour to its enclosure shape that resembles a cloud.

In the rest of this document we will explain all the hardware and software details of Cloudy, so that you can replicate this project by your own.

Figure 1: Cloudy Umbrella Stand.

2 Prototype description

Our systems consists of five parts: the Cloudy Umbrella Stand (from now on called Cloudy for short), the Cloudy Server, a forecast web service located on the internet (forecast.io), one or more iBeacon devices to uniquely identify an user and web interface to configure reminders. Figure 2 shows how these components relate together.

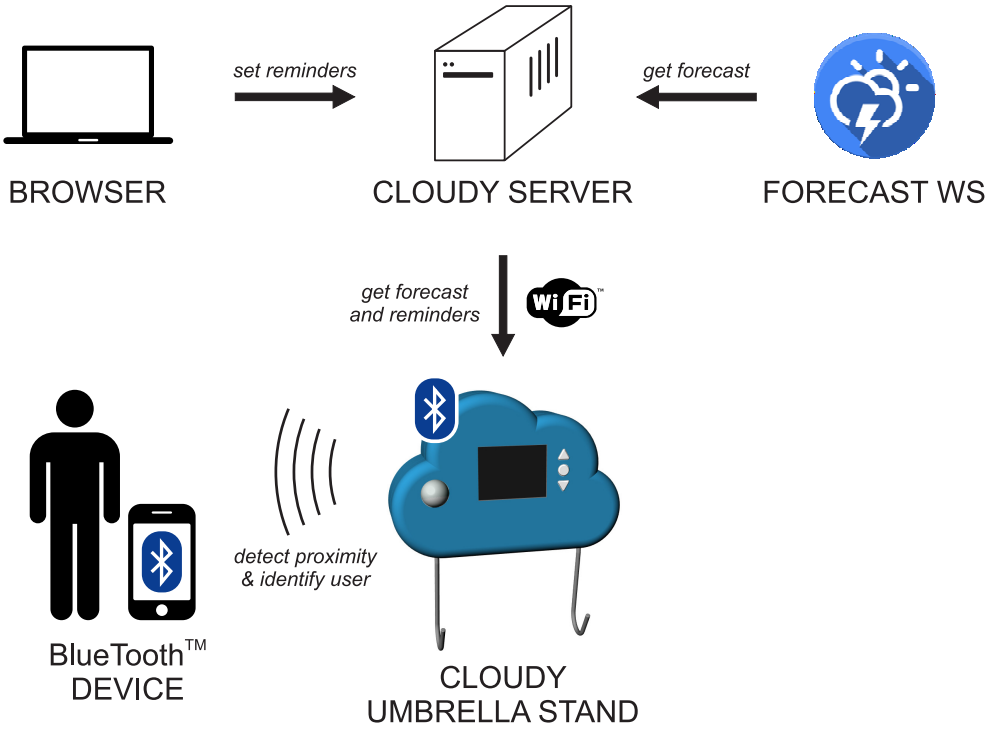


Figure 2: Cloudy components and their interactions.

Two typical usage scenario are considered, one for the registered user and one for the guest user.

The guest scenario is the following: the user pass in front of Cloudy, whose PIR sensor detects the presence of a person and activate the device. One call to the server is made to retrieve the up-to-date forecast for the current day. The server respond to this request either providing a cached forecast or getting the latest forecast directly from the web service, depending on the age of the cached forecast. Once the forecast are received by Cloudy, it displays it using the TFT display. Figure 1 on page 2 shows an example of the provided forecast. In the meanwhile, Cloudy scans for active BlueTooth[1] devices in the range of 1 meter and sends to the server a list of UUIDs of devices discovered: the use of these UUIDs is explained in the next scenario.

The registered scenario differs from the previous one for the capability of an user to create a personal profile associated with its BlueTooth device and

[1]At this time, Cloudy can discover only active iBeacon™ devices that periodically emits a signal. Modern smartphones can act as an iBeacon™ device, as well as dedicated devices like the Estimote™ ones.

to configure one or more reminders. Let's see how.

Using the browser, the user loads the page provided by the server and creates a new user profile. A list of detected BlueTooth UUIDs is presented, to let the user associate its device with the created profile. Then he can start to add one or more reminder. Each reminder can be associated with one or more of the following:

- a specific interval of time during the day;
- a weather condition: the reminder will be displayed only if the actual weather satisfy the condition (e.g. only when it will be a rainy day);
- an hook in Cloudy: in this case Cloudy will display the reminder only if something is attached to the specified hook. As an example, this enables the user to be notified only if his umbrella is attached to the hook, meaning that he is forgetting to take it.

If a reminder is not associated with an hook or with a weather condition it acts as a general reminder: an user can configure it to remember about important things to take with him. Launch, documents or keys are examples of stuff that one doesn't want to forget at home!

Figure 3: Example of a reminder configuration.

3 Cloudy Hardware and Software

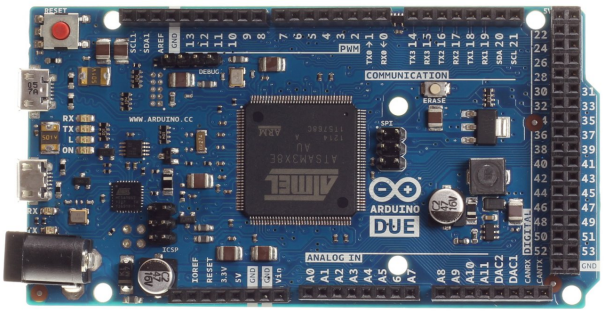
In this section we will describe the hardware and software of the Cloudy system. We will start analyzing the Cloudy hardware along with its embedded software, then we will look at the sever and its interaction with the forecast web services. The level of details should be enough for an user to replicate the entire project or adapt parts of it for its own purposes.

3.1 Cloudy hardware

List of components

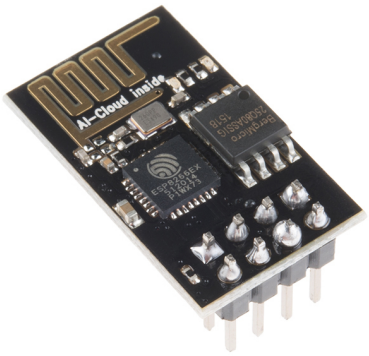
For the number of components and its communication capabilities, Cloudy can be considered as a small computer. Here it is the full list of components embedded into it:

- An Arduino Due. This powerful 32-bit ARM board has been chosen for the great number of its I/O pins and the size of the RAM, that allows to load very complex and large programs.

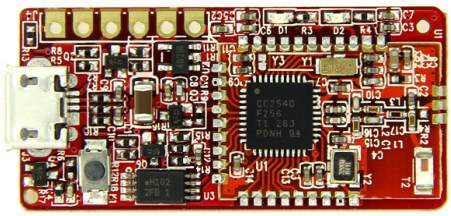


- An Adafruit 2.8" Color TFT display. This true color 240x320 pixel display is very versatile and easy to program. It can hold a microSD card behind it, to store image that can be successively drawed on screen. We connected it to the Arduino though the SPI interface.

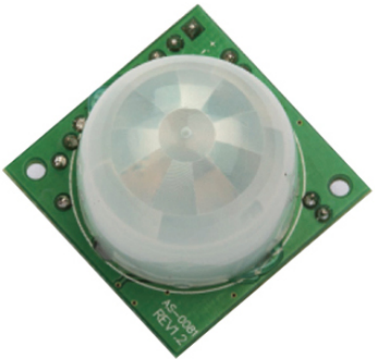
- An ESP8266 wifi board. The ESP8266 is more than a simple wifi shield, it is a complete system-on-a-chip that provides great programmability. It communicates with other devices using a standard serial interface and can be programmed in many ways: we chose to use the specific Arduino ESP8266 module, that allows to program the ESP in the same way as the Arduino itself.



- A RED Bear BLE mini BlueTooth shield. This board were chosen because of its capabilities to act as a *central role device*, so that it can operate without the need to be paired with another device. It communicates with the arduino via a standard serial interface.



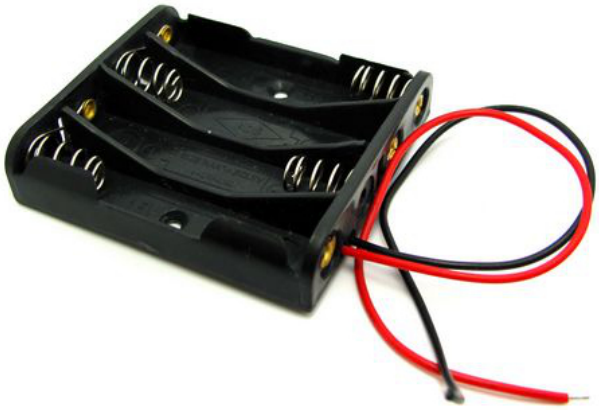
- A PIR sensor. A Passive Infrared Sensor is used to detect the presence of a person in front of Cloudy. It is connected to one of the Arduino input pins, raising the signal when something in front of it moves.



- A 3.3v piezo buzzer. Used to emit a sound and attract the attention of the user when a reminder is signaled.

- Three standard pushbuttons. Used for the navigations buttons: up, down, ok.

- A 4xAA battery holder.



- Some coloured cables, two small stripboards, connectors, screws.

The BLE mini firmware

In order to use the RedBear BLE mini shield in central role mode, we had to upgrade its firmware to the latest version, called "Biscuit". We followed the instructions on the redbear site: `redbearlab.com/blemini`. Here we want to warn you about two important things that must be considered when upgrading the firmware:

- a windows PC is needed;
- don't try to download the old firmware from the shiled to the PC, this doesn't work, only uploading is supported.

Programming the ESP8266

Due to its popularity and low cost, there are plenty of web tutorial on the internet explaining how to program the ESP8266 and how to use it for various applications. A detailed description of how to program it is beyond the scope of this report, but, based on our experience, we recommend the use of the Arduino ESP8266 module, which is a type of board supported by the arduino IDE as an extension. This allows to program the ESP8266 within the arduino IDE itself, with the same language as the arduino.

The Cloudy case

We wanted our case to remind its principal concepts, umbrellas and weather, so we gave it the shape of a cloud.

Designing and prototyping the case for Cloudy was a big challenge for the following aspects:

- the number of components to fit into it;
- the internal space that became quite cluttered;
- the need to place two hooks with the relative sensors for detecting the presence of something attached to them.

To minimize the number of prototypes to be printed, we decided to design both the case and all the internal components in 3D, to have the possibility of playing with the internal positioning of the components to find the best arrangement. This took quite a long time, but this approach rewarned us: only one single print of the case has been necessary. Since the total time of 3D printing for one copy of the case is more than 24 hours, this was a great time saving!

We designed all the case parts using the software Rhino 3D, a NURBS 3D software modeler widely used for prototype designing. Since in this case a picture worths more than one thousand words, refer to Figures 4 and 6 for more details of the case design.

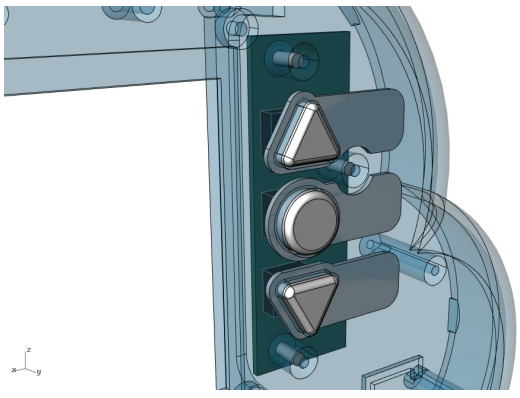


Figure 4: Design of the front buttons.

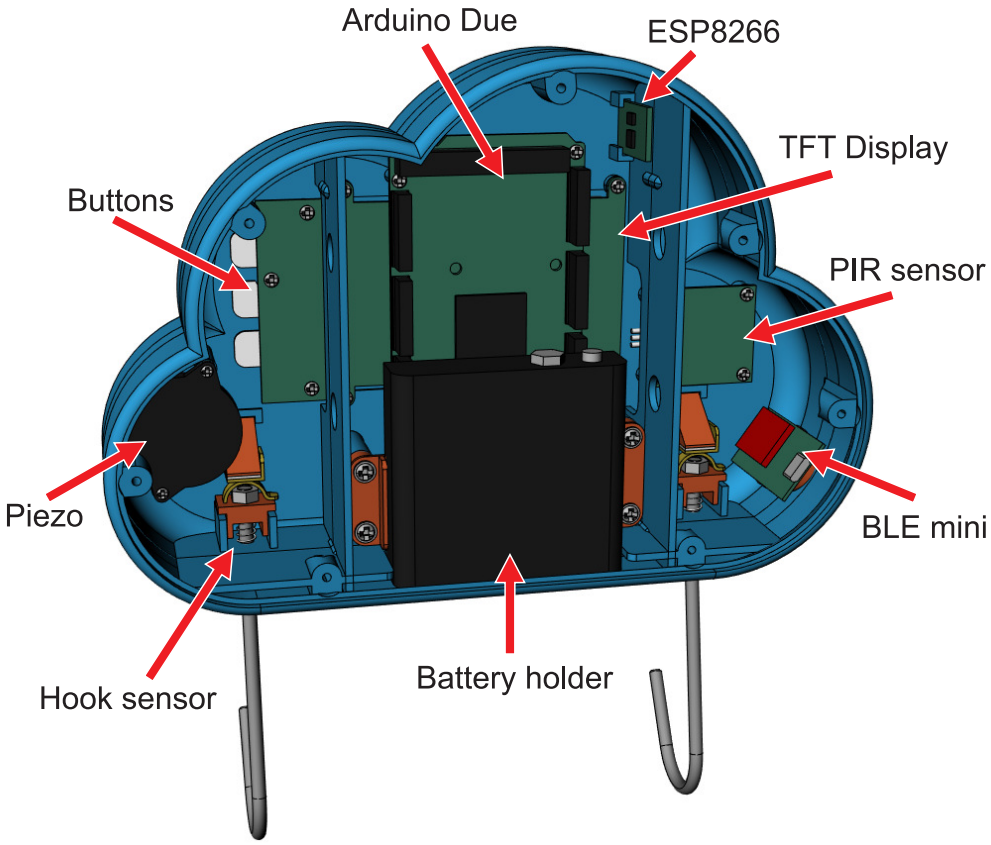


Figure 5: Placement of the internal components of Cloudy. Each component is fixed to the case with two or more screws.

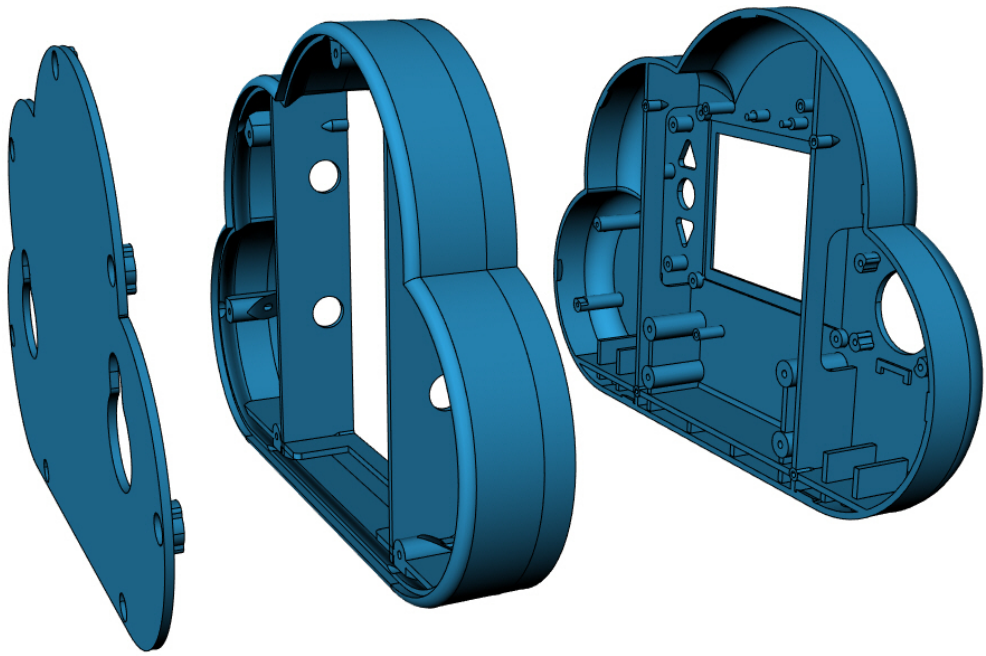


Figure 6: The Cloudy case. The case is splitted into three parts to ease the printing and the assembling phase.

Hook presence sensor

One of the most critical part of Cloudy is the sensor that detects, for each hook, if something is attached to it. The sensor was the first part developed, and it was prototyped separately. The final solution is built around a brass contact that is kept closed by a spring: one end of the contact is grounded, while the other is connected to a PULL_UP input pin of the Arduino. When something is attched to the hook, the spring is pushed and the circuit become opened, so that Arduino can detect the change on its PULL_UP pin.

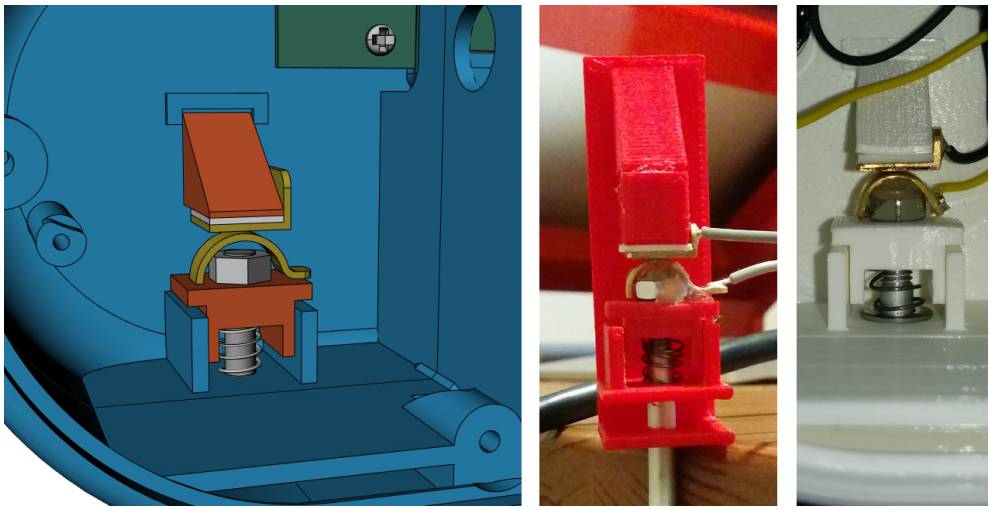


Figure 7: The hook sensor. On the left the final design. In the middle the first prototype. On the right the final version.

Wirings

Wiring up all the components is not difficult, every component pin should be connected to the corresponding pin in the Arduino without complex circuits.

The TFT screen communicates thought the SPI interface, whose pins are in a dedicated socket in the center of arduino. The ESP8266 and the BLE mini communicate via standard serial interface. Other components are simply connected to one of the digital input pins. See Figure 8 for the wiring diagram.

3.2 Cloudy software

Writing the Arduino software was another big challenge of this project. All the embedded components neeeds to be properly coordinated to provide the overall functionality of Cloudy. This means that the software should check various inputs at every cycle and change its behaviour accordingly.

The software is decomposed in several modules, see Figure 9. Each module provides a specific functionality to the system and some of them, namely IBeaconDetector and DrawBitmap, are general purpose and can to be reused in other projects. Here is the list of modules with their provided functionalities:

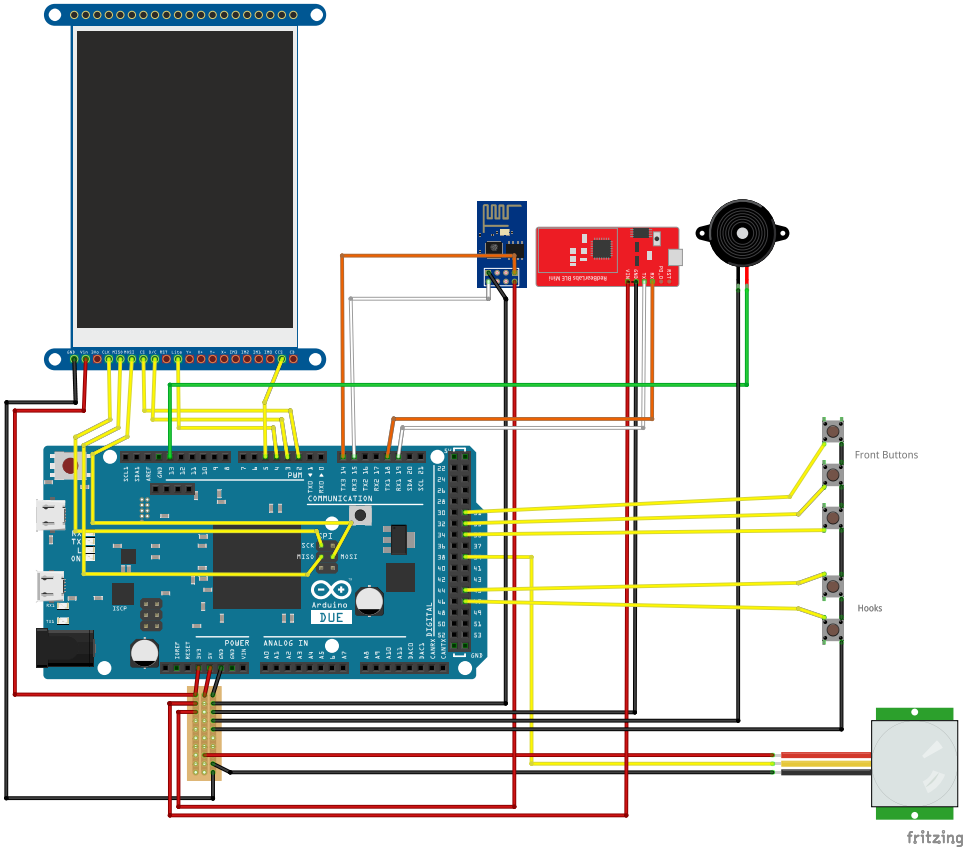


Figure 8: Cloudy wirings. The two hook sensors are modeled as buttons, but the are normally closed instead of normally open.

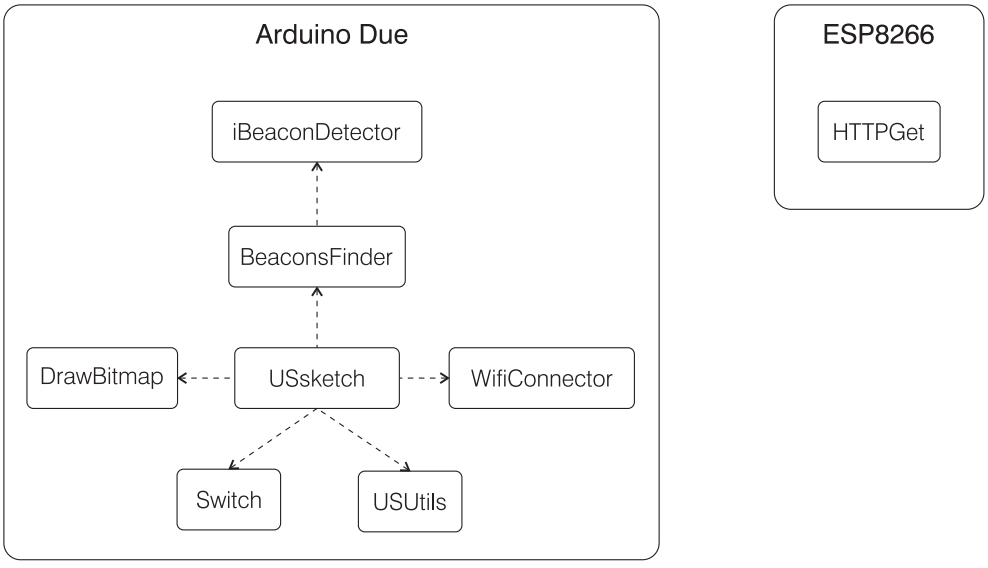


Figure 9: Modules of the Cloudy software. USsketch is the main orechestrator of the other modules. Dashed arrows show the dependencies. The module HTTPGet is deployed to the ESP8266 board.

- USSketch Is the main orchestrator of the system. Provides the initialization routine and the main control loop.
- iBeaconDetector Uses the BLE mini to discover active iBeacons devices. This module handles all the low-level details of the communications. Based on the work of Krishnaraj Varma.
- BeaconsFinder Uses the iBeaconDetector module to provide an higher level service, where iBeacons devices are discovered, filtered by distance and retained in a list for a given number of milliseconds.
- DrawBitmap Allows to draw arbitrary bitmaps files stored in the SD card onto the TFT screen.
- WifiConnector Communicates with the ESP8266 to provide wireless communications with the server.
- USUtils A collection of data structures and function to simplify the code in the USSketch module. Drawing routines for the TFT display are contained here.
- Switch Debouncer and deglitcher for input pins. This module is a work of Albert van Dalen.
- HTTPGet This module is deployed into the ESP8266 module. It provides HTTP GET communications with the server in order to retrieve JSON messages containing web forecast and reminders.

In order to compile the software two more libraries needs to be installed in the Arduino IDE, all availables from the Adafruit site:

- Adafruit_ILI9341-master
- Adafruit-GFX-Library-master

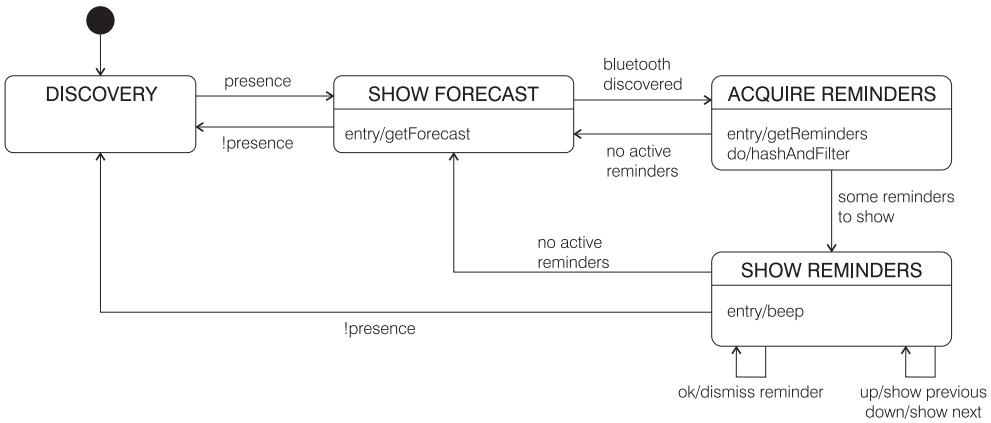


Figure 10: State diagram for Cloudy software.

Another important aspect of the Cloudy software that is important to analyze is its dynamic behaviour. Figure 10 shows a state diagram that summarize the main activities of the controller.

When in Discovery state, the controller look for PIR sensor to detect the presence of a person in fron of it, while at the same time scanning for BlueTooth devices in the range of 1 meter.

Once the presence of a person is sensed, the controllor goes in the Show Forecast state. Here an aupdate forecast is downloaded from the server and displayed into the TFT screen. If the presence of a BlueTooth device is discovered, then it goes to the Acquire Reminders state, otherwise it waits until the PIR sensor doesn't detect the presence of a person anymore and returns to the Discovery state.

In the Acquire reminders state a list of reminders associated to the detected BlueTooth devices is requested to the server and filtered. The filtering is necessary to account for the presence of objects attached to the hooks and to avoid the same reminder to be showed again if already dismissed. Based on the presence of active reminders the controller then goes to the Show reminders state or returns back to the Show forecast state.

Finally, in the Show reminders state the previously acquired list of active reminders is presented to the user. Using the three front buttons (up, down, ok) the user can scroll and read all the reminders and dismiss them pressing the central round button.

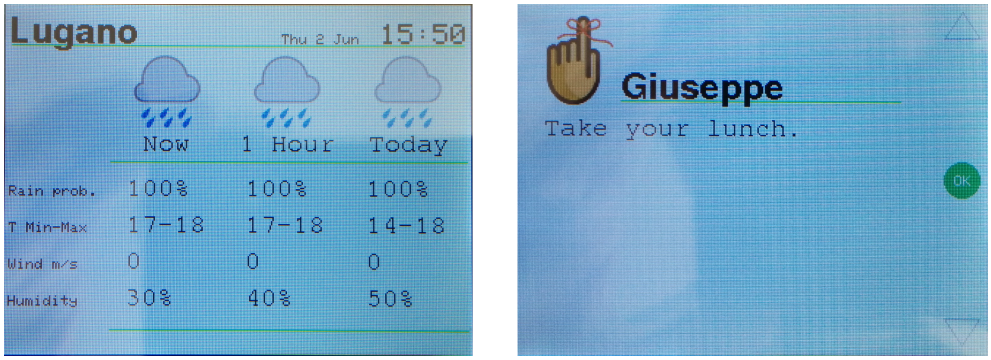


Figure 11: The Cloudy graphics: left screen shows the forecast, right screen shows a reminder.

3.3 The Cloudy server

In order to allow Cloudy to get information about the user's habits and forecast information, we needed to develop a Server.
The server is used for three different purposes:

- User profile web application
- Request weather forecast
- Serve Cloudy with needed data

User profile web application

When approaching the web service at the beginning, the user will be shown with a page asking him to login or signup. Both procedures, when completed, will respond with the user's profile page.

Figure 12: Index page: login or signup

Figure 13: Login page

On the profile page, the first time the user logs in, he has to register his

bluetooth UUID. Until he doesn't register it, the register bluetooth button will be available on the upper left.

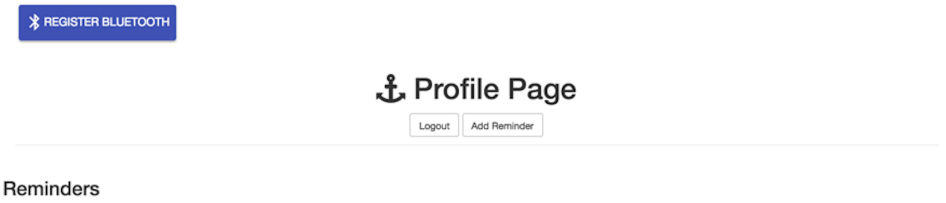


Figure 14: Profile page with index, with "Register Bluetooth" botton

By clicking on the Register Bluetooth button, the user gets redirected to a window containing the list of bluetooth UUIDs that Cloudy has seen but are not registered. Clicking on one of them will result in the association between user and bluetooth UUID.

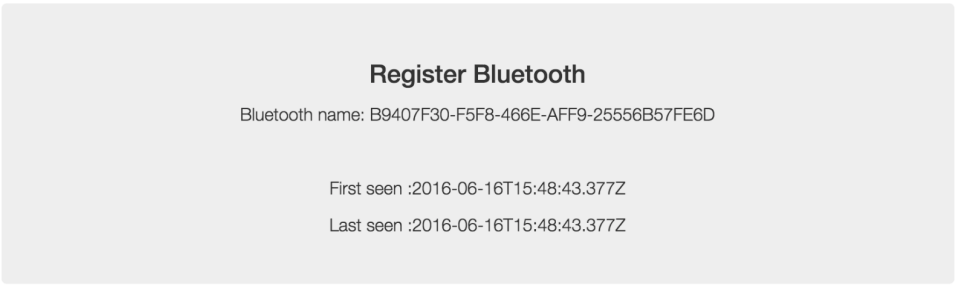


Figure 15: List of discovered unregistered bluetooth UUID

In the user's profile, the user will see the list of added reminders and a button for adding a new one.
When the user wants to add a new reminder, he will be served with a page where he is required to put a description of the reminder and add some constraints on it like for example the repetition days, the time span of when the reminder has to be shown, and if there is a particular weather condition in which the reminder has to be shown.

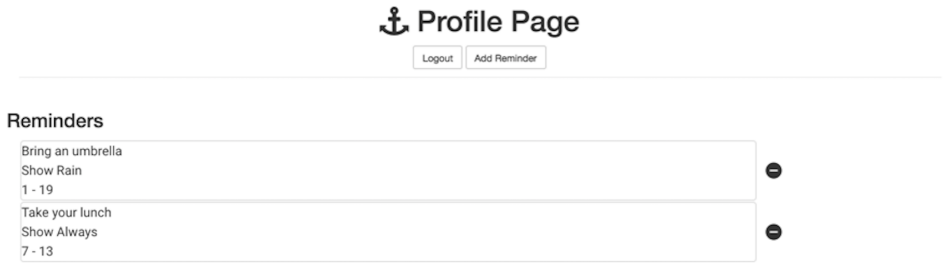


Figure 16: List of added reminders

Figure 17: Page for adding a new reminder

API

The server has a publicly exposed API in order to permit Cloudy to perform his requests for getting the weather forecast and the reminders.

Accessing the daily forecast is done by the url:
$http://\{serverIP\}:\{port\}/todayForecast$. This will return a JSON containing relevant forecast data about the hour it was requested, the next hour forecast, and the summary of the rest of the day's forecast.
When this operation is requested, the server first makes a request to forecast.io with the following URI:
$https://api.forecast.io/forecast/\{apiKey\}/\{latitude\},\{longitude\}\{?options\}$.
This request to forecast.io will return the forecast for the whole day in a JSON format, our server will filter the response by getting only the filtered data and make some average computation over wind speed, humidity, and precipitation probability. This data will be reformatted in a JSON object and sent to Cloudy.

Accessing the reminders is done by the url:
$http://\{serverIP\}:\{port\}/reminder/\{blID\}$. This request will return either a JSON Object as response or an information message. When Cloudy senses a bluetooth device, the request is triggered. As you can see in the flowchart in Figure 18, in the process we have two major decisions to take care of. The first one is if there is a user that paired with the bluetooth UUID that

was sensed. If this user exists, we get his reminders and check if the conditions hold (time span, day, weather). Instead in case there is no user associated with this UUID, we lookup if the bluetooth was seen before. If it was, we update the last seen field, in the other case we register it.

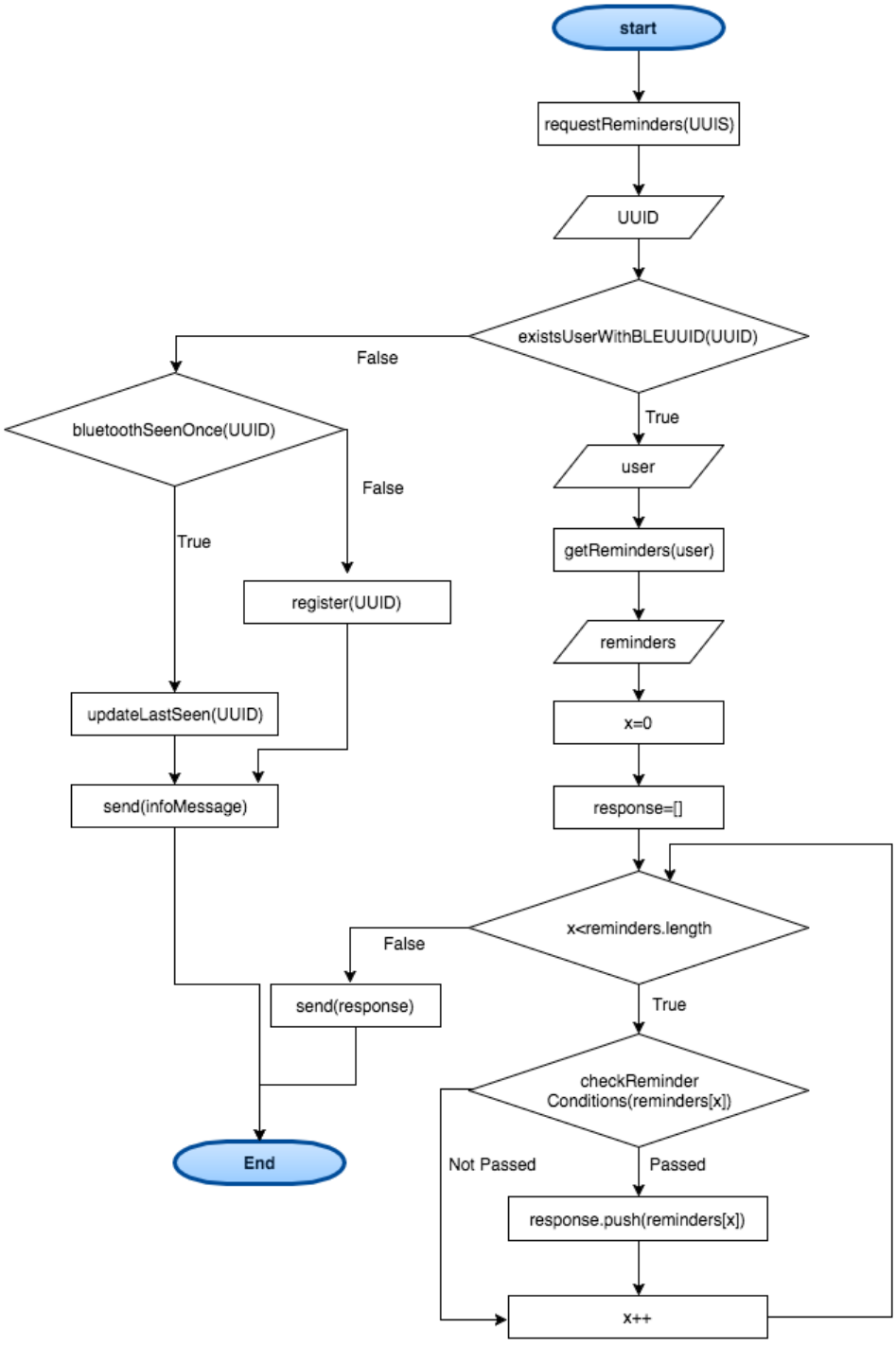


Figure 18: Flowchart depicting the process of the server when the reminders are requested

The Dark Sky Forecast API

The Forecast API allows you to look up the weather anywhere on the globe, returning (where available):

- Current conditions
- Minute-by-minute forecasts up to 1 hour
- Hour-by-hour forecasts up to 48 hours
- Day-by-day forecasts up to 7 days

`https://developer.forecast.io/docs/v2`, API Documentiation page
In order to be able to use this API an APIKEY is requested. It can be obtained for free under a certain number of daily requests by registering on this site. `https://developer.forecast.io/register`.
Once the APIKEY is obtained, it can be easily inserted under the variable apikey in a configuration found under the path WebInterface/forecast/params.js

Installation & dependencies

For getting the server started, it needs the Node.js platform, MongDB as a database for storing the data we need, npm for installing the server side dependencies and bower for installing the client side dependencies.

npm is the package manager for JavaScript.
Before starting the server we need to install all the modules required. For doing that we need npm (`https://docs.npmjs.com/cli/install`). After installing npm, we simply launch the comand *npm install* from the directory where the package.json resides. By doing so, the installation of the dependencies will start.

Bower is the package manager for the client-side web. In order to install bower after you installed npm, use the command *npm install -g bower* and then run the command *bower install* in the folder where bower.json resides. This will install all client side dependencies.

MongoDB is a document oriented database. We use this data base for storing the user profiles and data associated to them. Before starting the server, we need to install MongoDB first and then run an instance of it. For installing MongoDb refer to the this web site
`https://docs.mongodb.com/v3.0/installation/`.
After you installed MongoDb, run the comand *sudo mongod*.

Node.js is a cross-platform runtime environment for developing server-side Web applications. Now that we have everithing setup, we can start the server by using the command *node bin/www*.

Note For visualising the web site, use the **Chrome web browser** as we use components and dependencies that are not supported by every browser.

4 Results

When a user approaches Cloudy, the first interaction that happens is to show him the weather forecast. If the user is in possesion of a device capable of BLE advertisement, the UUID device is stored to Cloudy's database. After the first approach with Cloudy, the user can register to Cloudy's web service by inserting a username and a password. When he gets to his profile window, the user can associate a BLE UUID to his profile. The system will automatically assign him a hook if available.
In order to add some interactivity with Cloudy, the user can provide some reminders through the web application to be triggered when he leaves the house. Those reminders consist of a description and some conditions. A few examples could be:

- Remember to take your lunch, from 8.00 am to 12.00 am, from Monday to Friday
- Take the sledge, when snowing, from 8.00 am to 4.00 pm on Sundays
- Bring your tennis rackets, from 7.00 pm to 9.00 pm on Thursdays
- Take your sunglasses, when sunny, from 8.00 am to 4.00 pm, from Monday to Sunday

The second time the user will approach Cloudy, he will experience a complete interaction with it. Cloudy will display the weather forecast, his reminders if the conditions are satisfied, and if an umbrella in hanging on the hook, Cloudy will suggest the user to take it in case of future necessity on that day.

5 Final reflections

5.1 Development phases

The work was split in multiple phases. The first one was a requirement analysis phase, in which we defined possible answers to questions such as which main features should the umbrella holder have, what hardware components are needed and what can we do with them, but also how should the user interact with it. After that phase, we had a basic picture of the possibilities that were possible, so we could proceed with requesting the different materials and trying to work out the features we came up with. We begun by the creation of the 3D model for enclosing our hardware. Thanks to the 3D modelling and engineering skills present in the team, the case was studied carefully to fit perfectly all the hardware and each piece was precisely measured and thought where to place in the enclosure.
We could then continue and start working on the hardware part by programming the Arduino and connecting the components to it. In parallel, we worked also at the servers functionality. After the two parts were finished, we tried to let them communicate and we stumbled upon the major issue with this project that will be covered below.

5.2 Problems and issues

Bluetooth Discovery

One of the main issues we had at the beginning was the user device discovery. We first thought about having a bluetooth module that could discover all bluetooth enabled devices in his range field. This approach was not effective, as the active discovery mode is provided for very few models of bluetooth modules, and it is not guaranteed that the ones who possess it will work perfectly. Therefore, after a lot of research and trying out different bluetooth modules we had, our best candidate was the RED Bear BLE mini Bluetooth shield, that is capable to act as a central role device. Unfortunately, it can discovery only BLE devices capable of continuous advertisement, and very few devices available on the market are capable of this.

Cloudy-Server communication

Our biggest issue was the communication with the server. Basically, the problem, which is still existing, is that when the ESP8266 module makes a request, the response will sometimes not be complete. When the problem arises, Cloudy shows the last received response and not the newly received incomplete one. We were not able to fix this problem, but we think that a possible solution could be to try out different WiFi modules or even try some microcontroller with integrated WiFi unit.

6 Terminology

npm node package manager
BLE Bluetooth low energy
UUID universally unique identifier

7 Links

The software written for Cloudy can be found on the following github repository
`https://github.com/mendolag/UmbrellaHolder.git`