Sami Sainio - IoT project
December 2017

Hanna Haimakainen
Corentin Thomasset

REMOTE STOVE MONITORING

IoT Project - Metropolia University

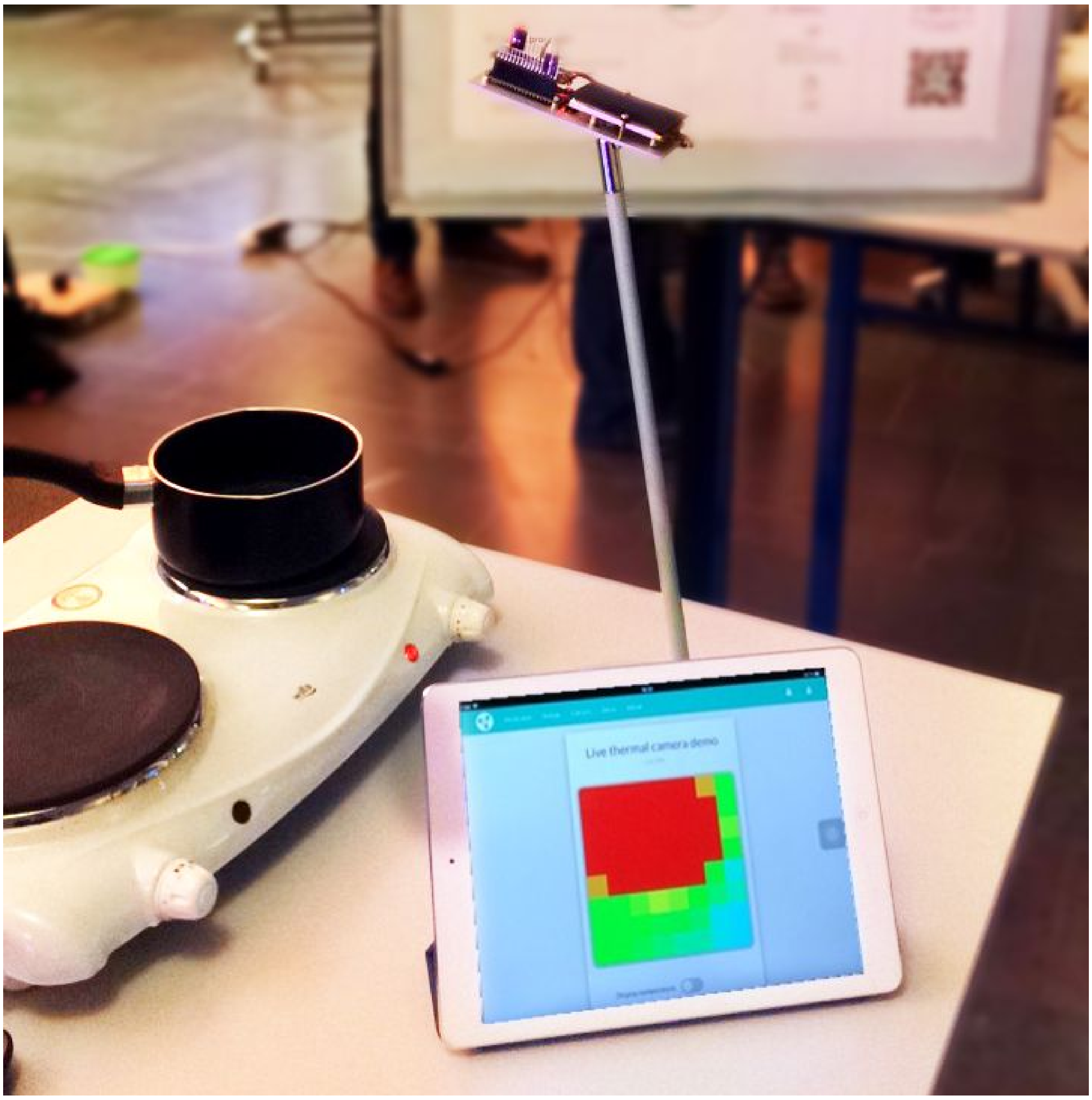


Table of contents

Presentation

Motivation & background:

During the years lived in shared apartments, authors noted the need for an automated stove top temperature monitoring as a preventive safety measure. Older conventional electric stove plates used in Finland are not equipped with safety turn-off feature, and it's highly possible to accidentally leave stove plates on after using the plates for making food. There are multitude of devices on the market based on movement detection sensors that are addressing this problem[1][2][3]. These systems track the duration of stove/user inactivity and can turn the stove automatically off after certain period. However, in shared student apartments making modifications to electrical stove connections can be prohibited[4]. Thus there could be a need for an aftermarket device that, even though if it can't physically control the stove, it could warn the user for left-on stove plates with some form of a warning message. This was our project's main goal: to engineer an internet-of-things -based easy to use warning system. A modern, web based graphical user interface was developed for this purpose with a small sensor device that could be attached after-market at the stove top.

Another interesting topic for stove top monitoring is the possibility for food temperature measurement. Information of the food temperature could be used to reduce wasted heat as the plate power settings could be adjusted more precisely and there would be no need to use highest temperature setting for just-in-case. Food temperature setting would also allow precise temperature control of the food as needed in some gourmet/delicate foods such as melting sugar for candies. Food temperature measurement was selected as a secondary goal.

Due to restrictions in modifying existing electrical stoves, we decided to use a contactless infrared measurement in both the stove activity monitoring and temperature detection. That way all the measurements could be done remotely without safety hazards. The following chapter describes the scientific background and reasoning behind the infrared measurement

[1] http://wallflower.com/
[2] https://inirv.com/
[3] https://iguardfire.com/
[4] There was also a prohibition to modify power electronics on this course

Infrared sensors

According to Planck's law[5] all matter radiates energy along the electromagnetic spectrum based on its physical temperature. For objects close to room temperature the peak of the distribution for radiated energy lies in the infrared wavelengths. This makes the sensor optimized to detect infrared radiation suitable for measuring object temperatures.
All infrared detectors can be classified into two groups: thermal and photonic [6]. Thermal detectors are designed to heat up from the measured infrared radiation, and this heat generates thermoelectric voltage (thermopiles) or changes the sensor resistance (bolometers). Bolometers and microbolometer array detectors are still expensive due to complicated fabrication. [7]

Photonic detectors take advantage of photoelectric effect, where absorbed infrared photon excites electrons in the sensor material and changes its resistance (photoconductive detector) or creates a voltage (photovoltaic detector). Photonic infrared detectors typically require active cooling system due to high noise, making them also expensive.

Thermopile sensor

In this project, we chose to use a thermopile infrared sensor due to low cost and complexity. Thermopile sensor consists of a series of interconnected thermocouples that generate a voltage based on temperature difference between two thermocouple junctions, with a thermally resistive layer between. [8][9] In infrared thermopile sensor, the other side of layer can be exposed to IR radiation through an infrared-passing filter, heating up that side of sensor. The temperature of the other side of sensor can be monitored with an ambient temperature sensor. (eg. NTC resistor)[10] . By measuring the voltage generated on thermocouples and taking account the ambient temperature, the temperature of IR-exposed side (and object temperature) can be computed.

[5] https://en.wikipedia.org/wiki/Plancks_law
[6] https://en.wikipedia.org/wiki/Infrared_detector
[7]http://www.laserfocusworld.com/articles/print/volume-48/issue-04/features/microbolometer-arrays-enable-uncooled-infrared-camera.html
[8] https://en.wikipedia.org/wiki/Thermopile
[9] https://www.sensorsmag.com/components/demystifying-thermopile-ir-temp-sensors
[10] TS305-11C55 Thermopile Sensor datasheet

Emissivity and reflectivity

Emissivity means the ability of matter to absorb and emit electromagnetic radiation, indicated as a percentage factor between 0.0 and 1.0 [11] In infrared measurements emissivity shows how accurately the temperature measured represents the actual temperature of the object. A room temperature object with high emissivity can be measured accurately, whereas the amount of IR radiation and thus the apparent temperature of a low emissivity object is much lower. Infrared detector cannot detect the emissivity of the object, making all measured results with low-emissivity objects inaccurate. Materials with good emissivity include water (0.95) and black surfaces (iron kettles) [12].

As food consists mostly of water, surface temperature of food can be measured accurately. However, surface temperature does not necessarily equate to the temperature inside the food, and regular stirring is needed to even out the temperature in the food if surface measurement is used.

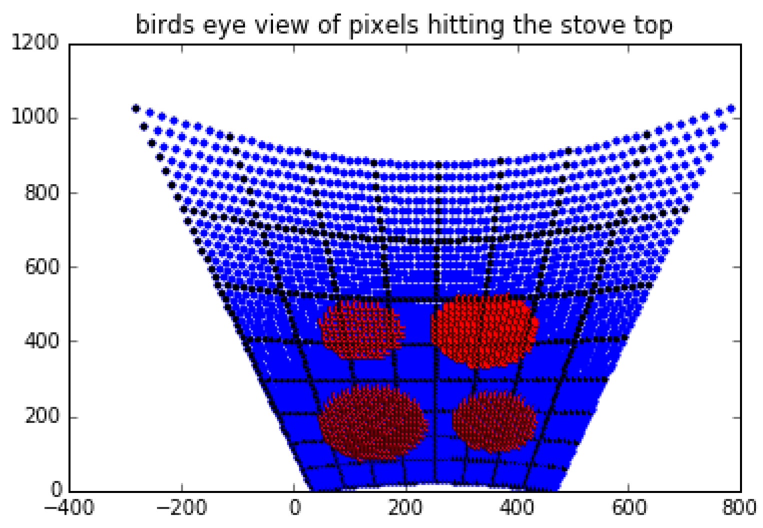
Temperature of polished metal pans/kettles can't be detected as they have really low emissivity. Opposite to emissivity, their reflectivity is high. Reflectivity defines the matter’s ability to reflect incoming radiation. Highly reflective objects don't emit or absorb much infrared radiation, but they reflect and bounce the infrared radiation from their surroundings, acting like a mirror. Visible, highly polished cooking ware will cause errors in stove top measurement due to reflections.

Camera analysis

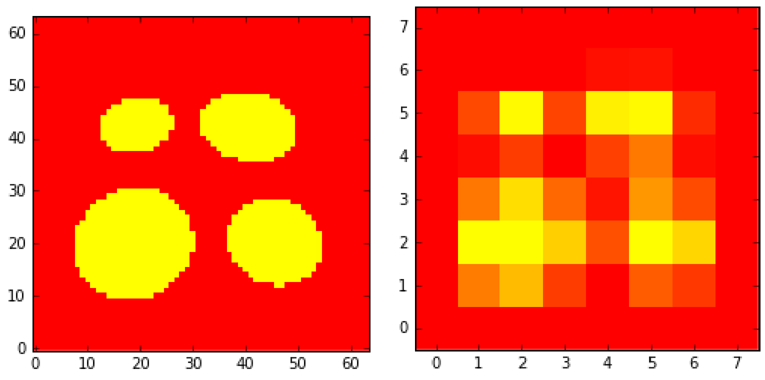
Requirements for the infrared sensor were discussed, the main requirement being the ability to detect stove plate temperatures separately. This required either multiple detectors or a single scannable detector. Scanning sensor was deemed to be too complex mechanically, and finally an AMG8833 Grid-Eye thermopile array sensor was selected as the sensor for the project[13] . Sensor consists of an integrated 8x8 array of thermopile detectors, with the whole combined field of view (FOV) being 60 degrees.

A physical placement for the sensor was designed to be either directly over stove top attached to the ventilation fume cover or attached to the wall behind the stove with sensor tilted towards the stove.

[11] https://en.wikipedia.org/wiki/Emissivity
[12] https://www.optotherm.com/emiss-table.htm
[13]https://industrial.panasonic.com/cdbs/www-data/pdf/ADI8000/ADI8000C59.pdf



A proof-of-concept simulation for camera image was written in python to prove that stove plates would be distinguishable in the final image. Simulation uses simple ray-tracing approach to track the intersection points of stove top plane and rays shot from individual pixels of camera array. Each ray was given a temperature depending on if it intersected with a stove plate or not. Resulting temperature and intensity of a single pixel in the simulated camera image was averaged over multiple sub-rays.

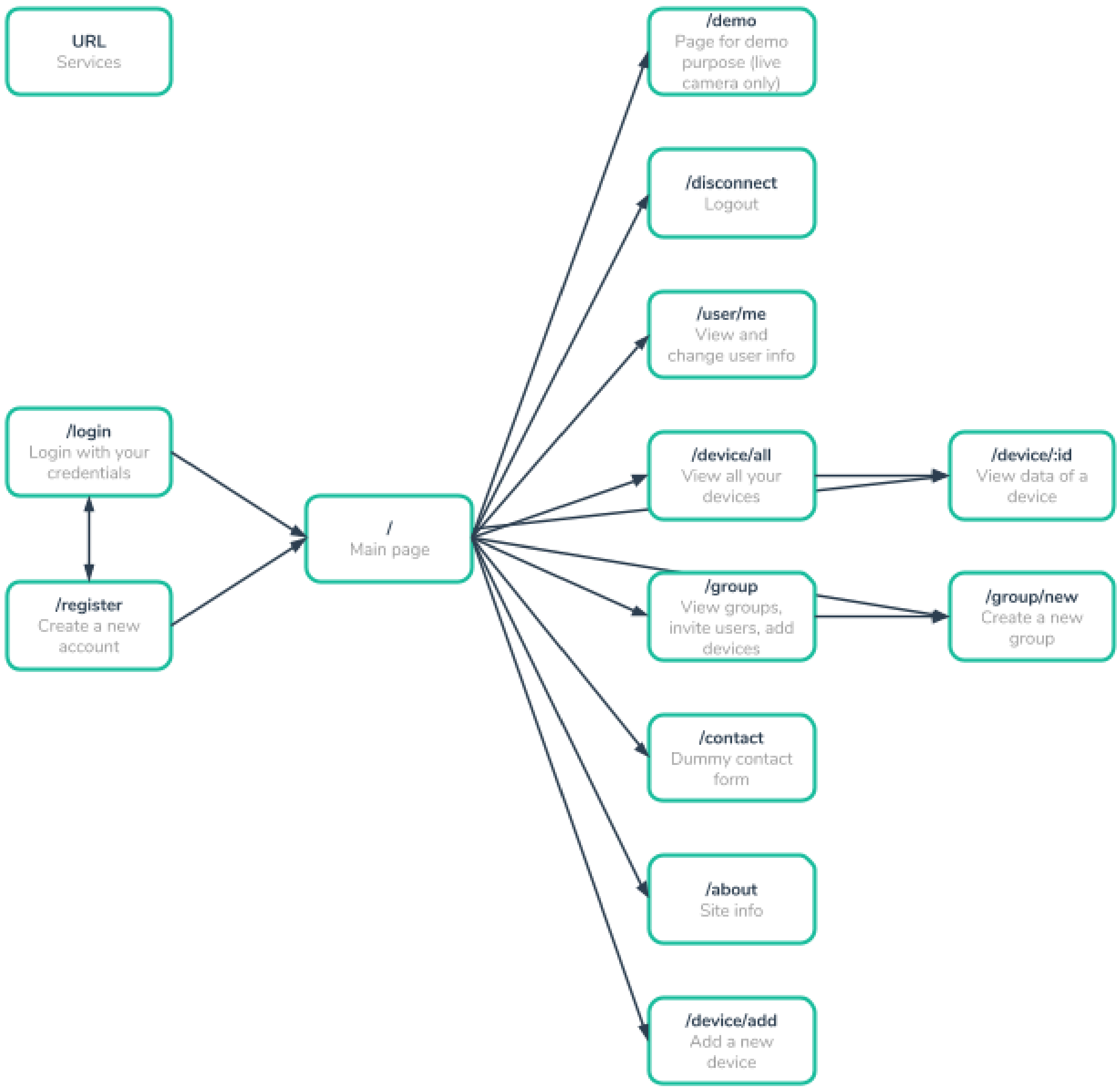


Simulation showed that all stove plates were distinguishable from each other at the distance of 60cm over the stove top. The apparent size of plates were more than single pixel each, which showed that plate temperature could be measured accurately. If the apparent size of plate in the image were only one pixel, it would be possible that plate size was smaller than a single pixel, part of the pixel temperature coming from plate and part from surrounding stove top diminishing the resulting temperature.

Simulations showed up the effect of perspective distortions in the image with tilted sensor. The problem was discussed, but not deemed of importance for this project.

Functionalities

Our application has been thought as a final product for an IOT company. It embeds an user management system: register, login, change password and email. Each user can add devices to it's account using a small serial number that may be printed on the device. Groups could also be created in order to allow several user to share devices data with other users (eg: for a family, flatmates). Our application can be tested here.



Sitemap of our application

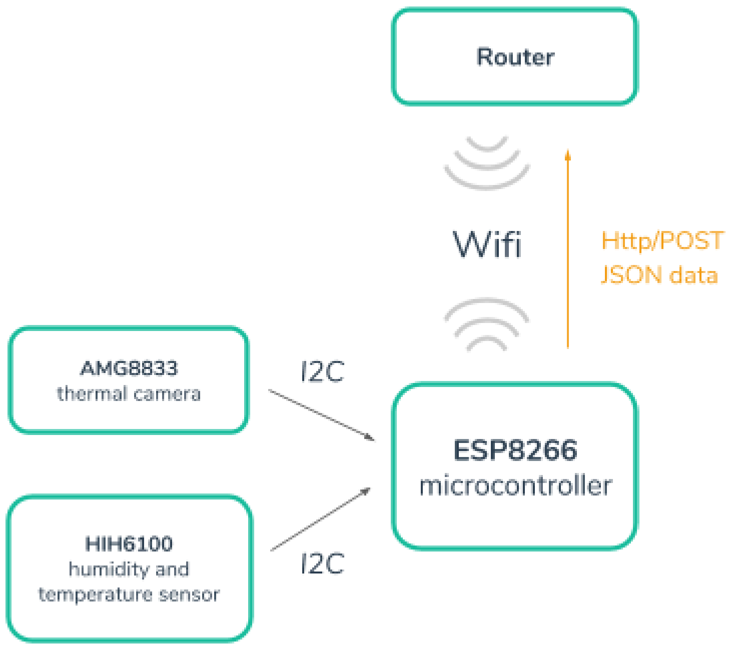
Using the application you can have all the following information:

- The live state of the device (ON/OFF)
- The current temperature and humidity of your kitchen and a graph of the evolution (updated every 10 sec)
- The amount of time between now and the last data reception
- A 8 by 8 pixels live thermal view of your kitchen plates
- A schematic of your actual detected plates and their temperature
- The evolution of the temperature of each plate
- Device configuration interface (change name, allow/disallow notifications)
- Device notification interval, configure intervals in which your plates should be turned off.
- Basic user management system
  - login
  - register
  - my account
  - change password
  - change email
- The group management system
  - Create a groupe
  - Invite a user
  - Add a device
- A live notification system

Screenshots and detailed explanation of each view can be found in annex 1.

Technical description

The device



The sensor device consists of a wifi-equipped ESP8266 microcontroller, an AMG8833 thermopile array thermal camera for measuring the plate temperatures and HIH6100 humidity and temperature sensor for ambient measurement. Microcontroller uses single I2C bus to read both the sensors.

Firmware

For simplicity, the device uses open source Arduino core as a base for the microcontroller firmware. Arduino project has a large open source user base and ecosystem, which is used for advantage in this project. The communication with thermal camera is also done with a separate open source driver[14]. The implementation of system software is minimalistic: device is only reading sensors and sending data to server. The main program consists of a simple state machine that has three states:

Normal mode

In normal operation, the device reads in the data of both sensors through I2C bus, encodes the data into a JSON-formatted string and sends it to the cloud server

[14] https://github.com/adafruit/Adafruit_AMG88xx

inside a HTTP POST request with a timestamp and device identification token. There are two different types of messages that are sent at different rates: full data packet with camera data and ambient sensor readings sent every 10s and bare live camera feed sent at approximately 3 FPS. No data encryption was used in the current design.

Setup mode

Device tries to connect automatically to home wifi network during startup using flash-stored access credentials. If connection fails, device creates its own connectable wifi access point with a web page where user can enter the network credentials. Network setup uses open source *Wifi Connection Manager* project[15]. After successful update of settings, device stores the settings, connects automatically to home network and enters normal operation mode. Setup mode can be manually triggered by pressing the onboard switch of the device for 20 seconds.

Reset mode

If onboard switch is pressed for over 5 seconds, device reboots. Device remembers its current settings.

Additional components

NTP Client

A dedicated NTPClient class was written to prepare and manage timestamp information.. At the start of the program NTPClient object holds the program execution until a valid timestamp is received from a Network Time Protocol server. After a valid fix, NTPClient enters to an internal state machine, which governs how the timestamp is computed. When user requests a new timestamp, new timestamp is computed as a sum of the last received NTP timestamp and the internal system millisecond timer. Object tracks internally the age of the stored NTP timestamp, and if saved timestamp is too old, requests a new NTP timestamp from network time server.

HIHSensor

A simple I2C driver was written to read temperature and humidity data from HIH 8100-series sensors. The driver initiates connection I2C connection to device address and reads out 4 bytes of data: 2 bytes of humidity information and 2 bytes of temperature information.

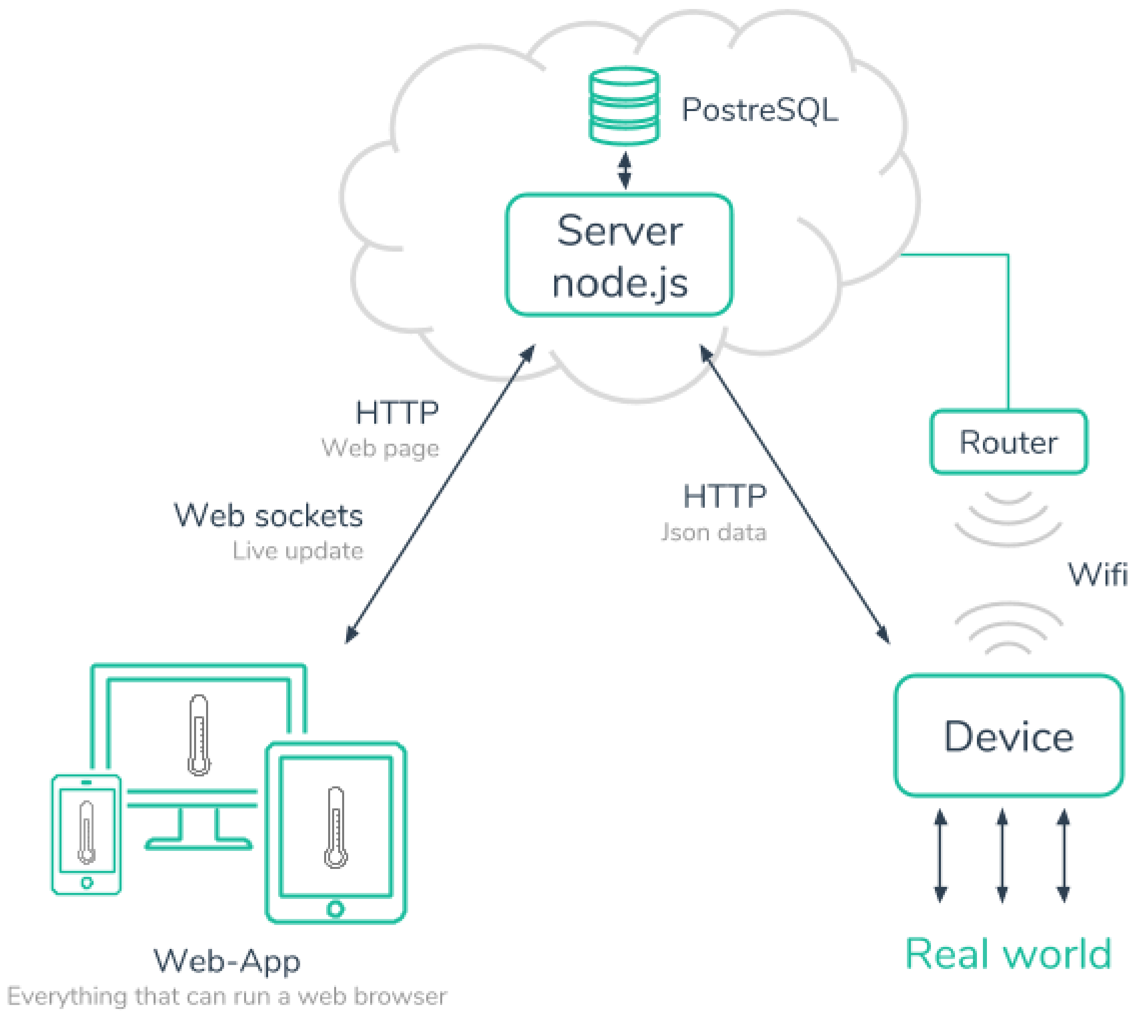
[15] https://github.com/tzapu/WiFiManager

The server

We built a node js server, hosted on Heroku during the production phase. This server has 2 main goals: serve as a web server and as an interface for the device.

The device will send HTTP request to special URL of the server to send data for database insertion and live views.

In order to have live updates of the layout and to make the user feel more comfortable with it's experience of the application we use web sockets. Web sockets permits to have asynchronous communication between the frontend and the backend. So don't need to refresh the page to update some part of the layout of our application. Our web-sockets communication is implemented using the Node.js package Socket.io.



Architecture of the project

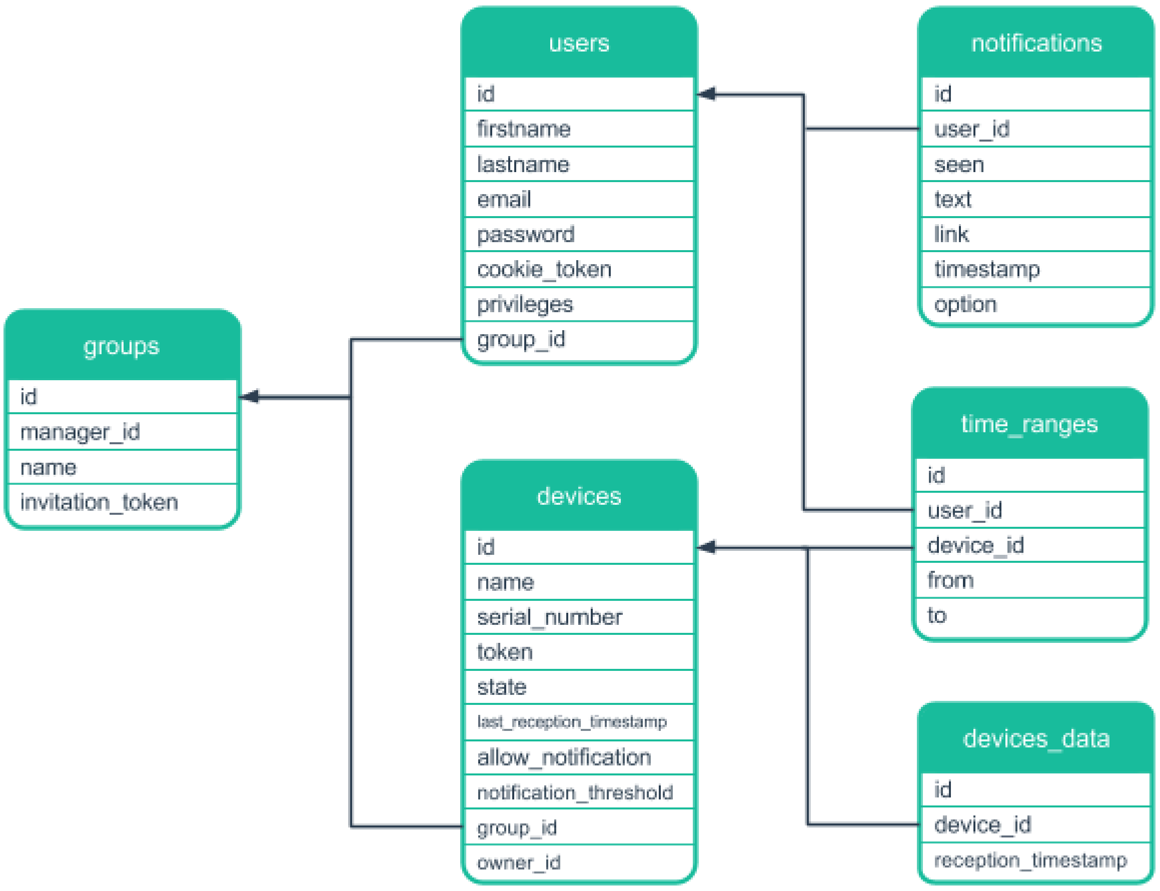
The database

In order to quickly test our product and to be sure to have a full control of the database and the communication protocols we chose to make our own backend solution.

During the development phase we used a SQLite[16] database because it's easy to debug and maintain. We wanted to keep this system for production state (on the Heroku server) but Heroku doesn't allow such kind a database. So we switched to a server based one: a PostgreSQL database hosted by Amazon AWS and provided as an Heroku add-on. So in the end of the project we had to migrate the SQLite database to the PostgreSQL one and make our application compatible with such database.

I decided to use a SQL based database because it's something that I know and since we built our own backend from scratch I wanted to use something quick that doesn't require us to send many time learning all subtleties of other type of data storage, such as MongoDb.

Structure of the database:

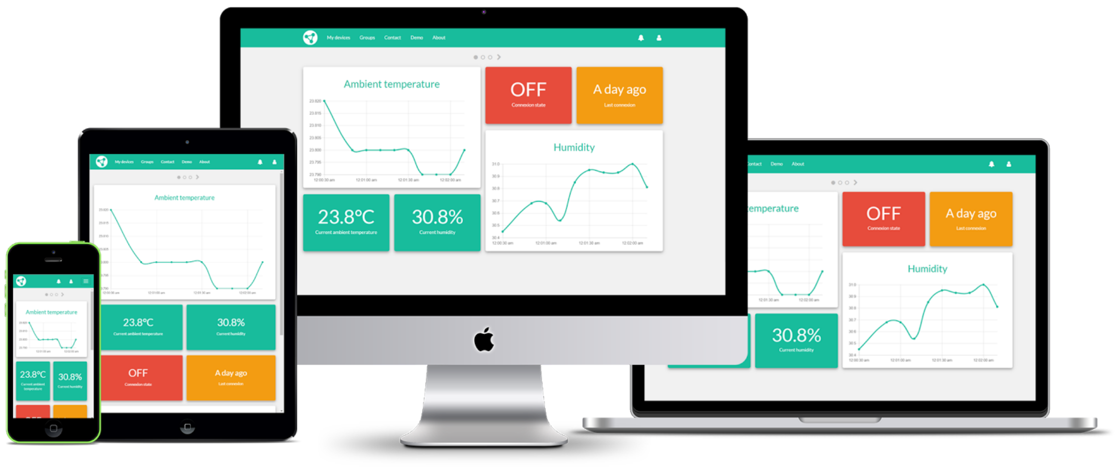


(You can find the details of each field in annex 2)

[16] SQLite is not a client–server database engine. Rather, it is embedded into the program as a file.

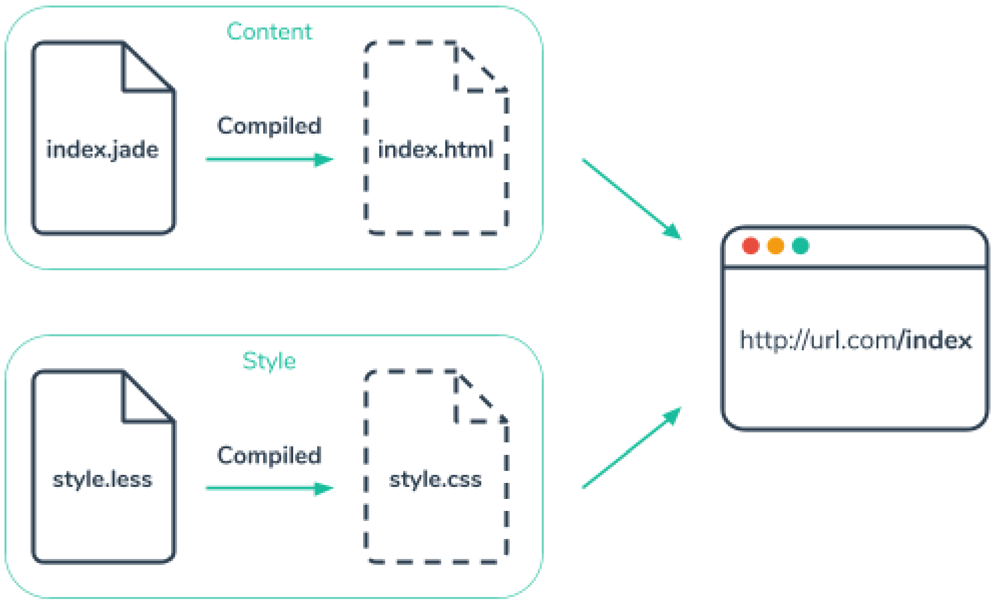
The application

Our application is a web application, basically a web page. The web page is completely responsive in order to be scale perfectly on every type of device. So the app is adapted to be use with a phone, a tablet or a browser. And in the other hand, our application is not platform dependant: it could be used on android, or IOS for example. Basically everything that has a web browser can use our services.

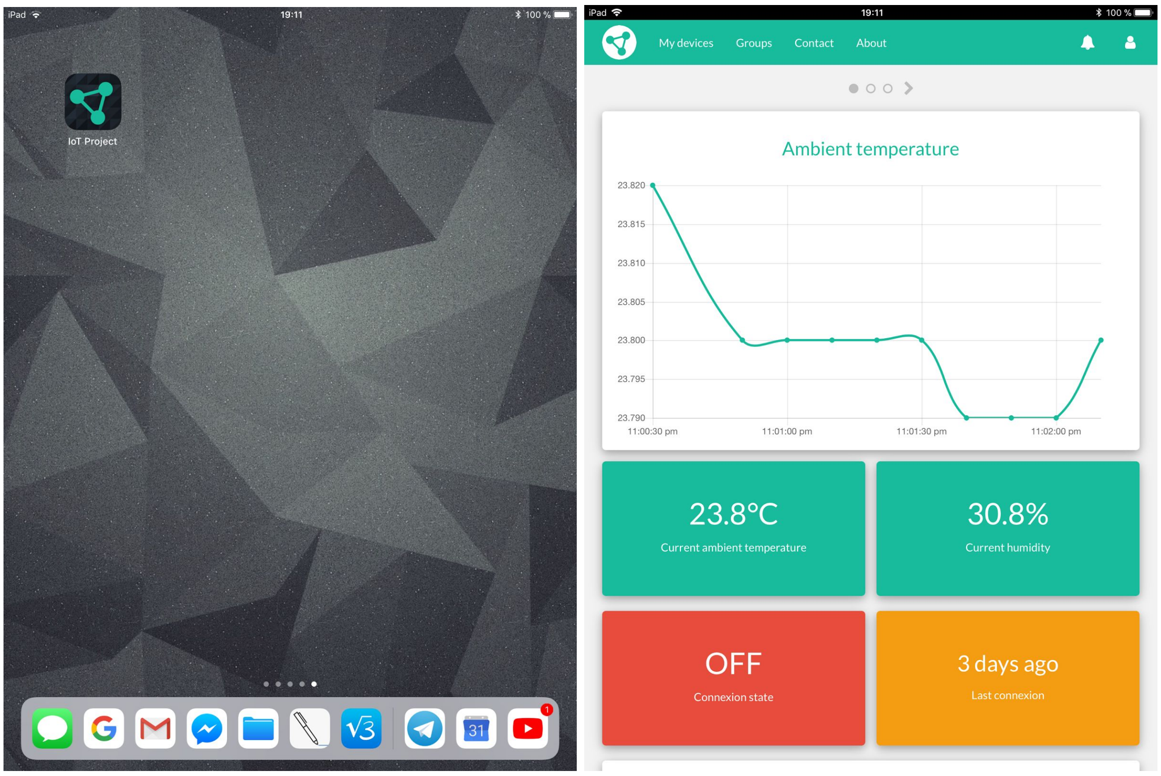


Responsiveness of our web app.

The layout is basically made of HTML and CSS. In order to ease the development part I used Jade, an HTML template engine (to quickly build the web pages) and LESS, a higher level language compiled into CSS.



Also, thanks to IOS 11, you can now run webpages as an application. This is call a standalone website, with some basic configuration in the header of the webpage we can easily set an icon and a name for the application. By using such kind of application you can run a web app and get rid of some browser features (such has the navigation bar and stuff like that). So it's just a website disguised as a native application.



The app in standalone mode on an Ipad

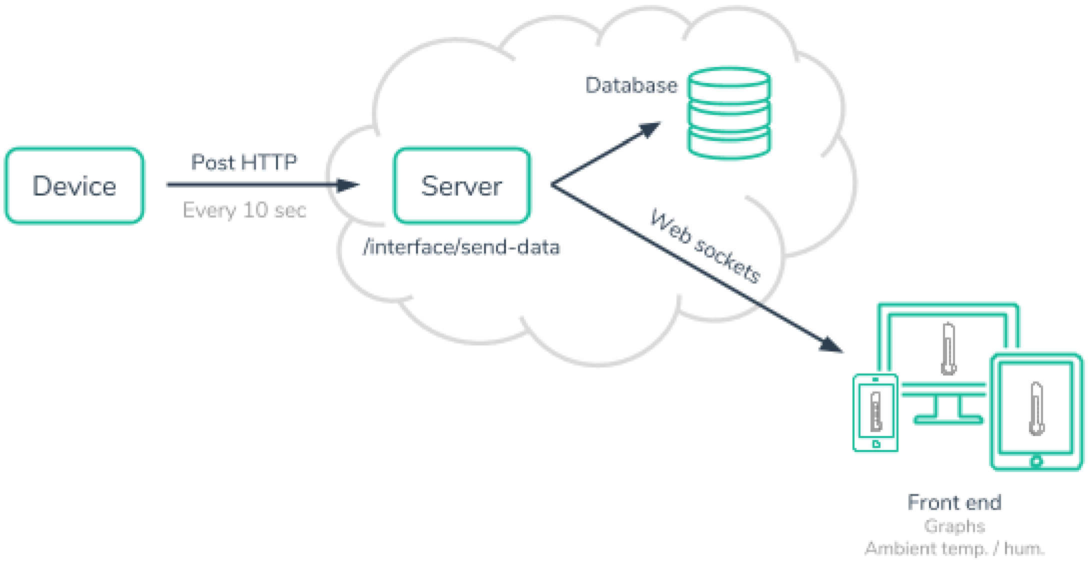
Device interface

In order to have the device to interact with the application and the database we created a set of URL to which the device can post HTTP request with all needed informations.

Data are formatted in JSON by the device and send to 2 different URL:

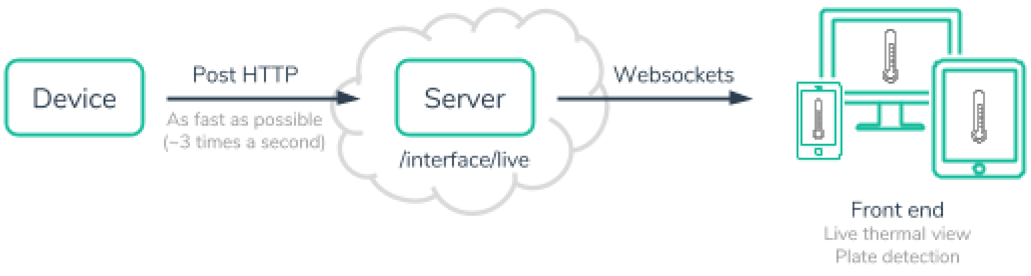
- **Standard behaviour**: http://serverurl.com**/interface/send-data**.

The data that are sent to this URL are stored in the database and send to front end.



- **Live update**: http://serverurl.com**/interface/live**.

The data that are sent to this URL are immediately send to the front end.



Issues encountered

Database migration

Switching from the SQLite to the PostgreSQL database was not something we had planned to do, we wanted to keep the SQLite one. But since Heroku doesn't allow such kind of database, the SQLite was wipe clean every 24h.

So we were obligated to find an alternative and we find out that Heroku can provide a PostgreSQL database has an add-on to the project. So we created one and migrated all the content of our previous database.

the most time taking part was to adapt the code of our application so that it become compatible with the new database.

HTTP POST requests

Initially inserting additional data payload to HTTP POST request didn't work with our client device and server. Issue was resolved by rigorous debugging and adjusting the formatting of POST request to adhere better to the standard.

Future prospects

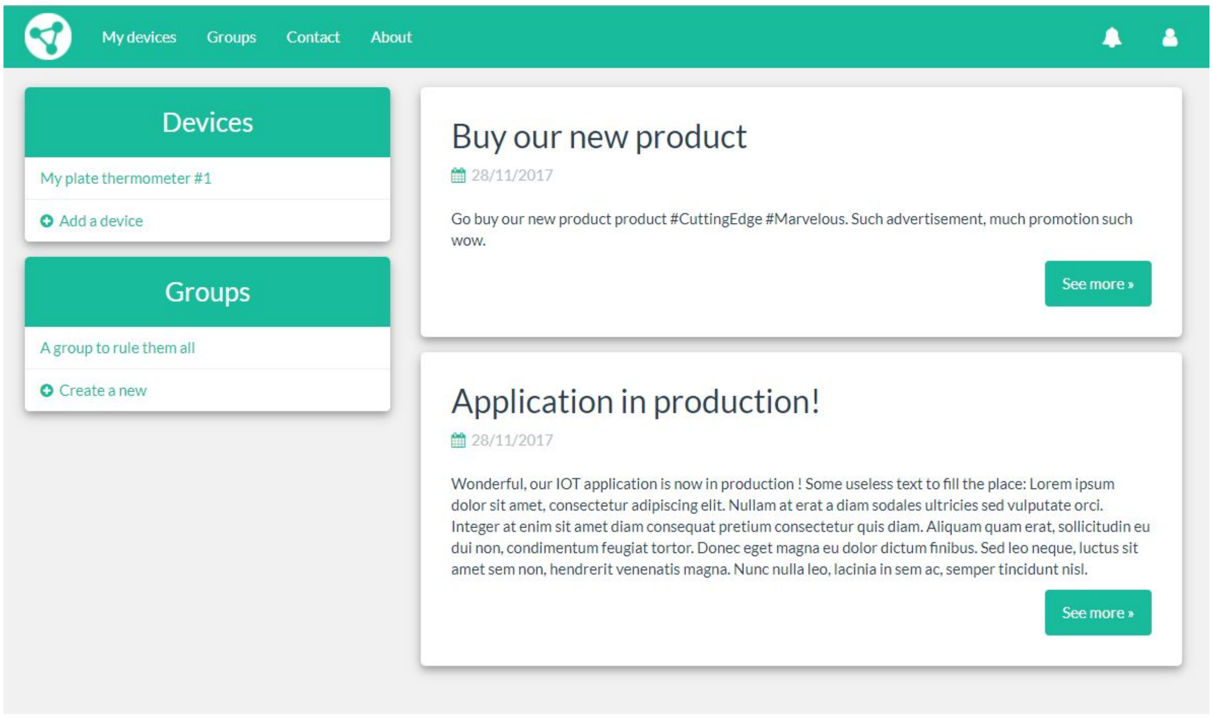
There are some areas that could be improved in the current design. The sensor could be integrated with the stove system, making it possible to control plate temperatures by using heat sensor data to adjust the input power to plates. This would show up in energy savings as there would be less plate overheating. Plate temperature control would make it possible to control temperature of foods with really high water content like soups. Food needs to be stirred before measurement to even up the temperature differences, though.

Plate lifetime could be monitored by measuring the heating plate current and the apparent plate temperature. Bad current-use-to-temperature ratio could show heating plate wear or need of cleaning.

The developed server backend - frontend design is a good platform for new products on its own, and could be repurposed to new use cases.

Annexes

1) Screenshots of our application



Buy our new product

28/11/2017

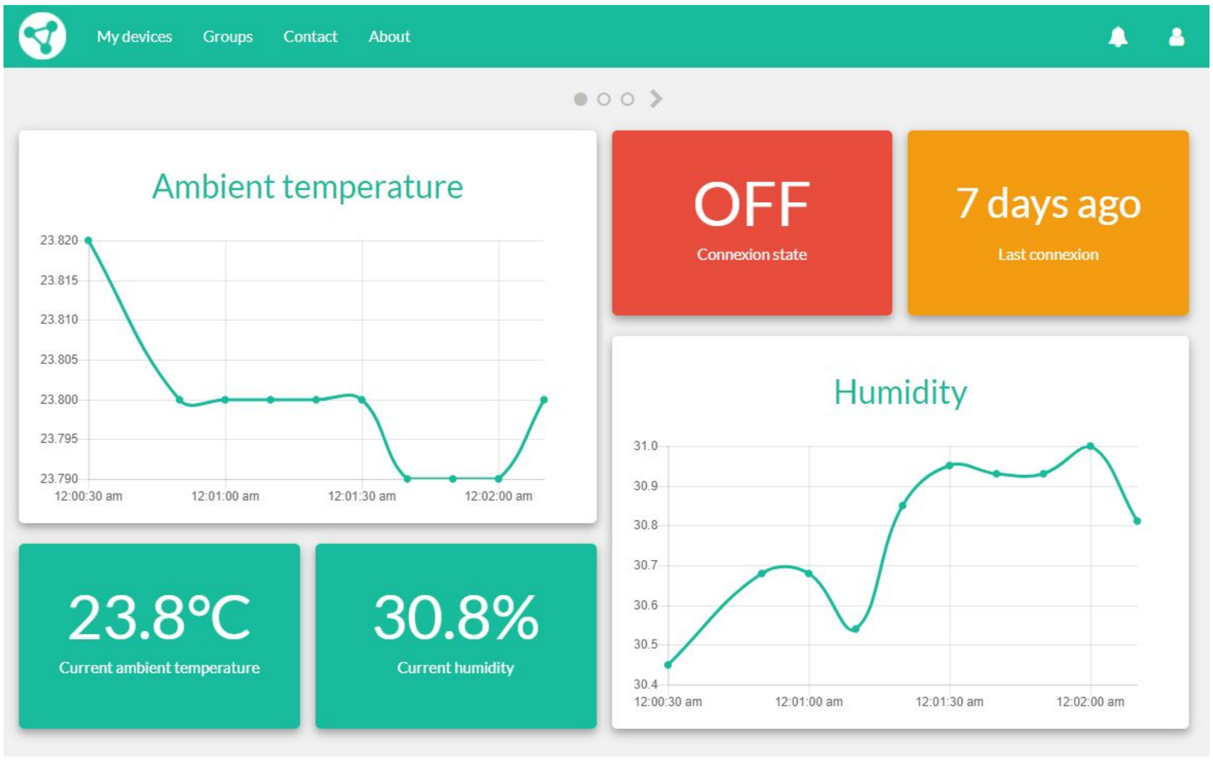
Go buy our new product product #CuttingEdge #Marvelous. Such advertisement, much promotion such wow.

Application in production!

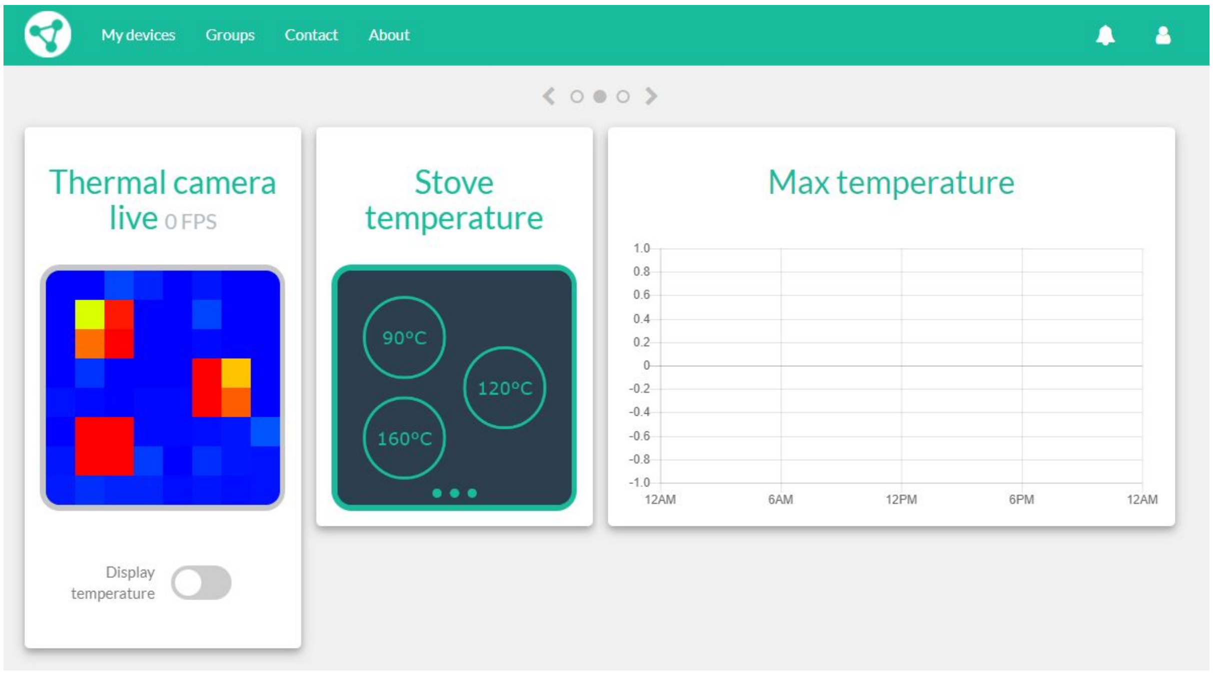
28/11/2017

Wonderful, our IOT application is now in production ! Some useless text to fill the place: Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nullam at erat a diam sodales ultricies sed vulputate orci. Integer at enim sit amet diam consequat pretium consectetur quis diam. Aliquam quam erat, sollicitudin eu dui non, condimentum feugiat tortor. Donec eget magna eu dolor dictum finibus. Sed leo neque, luctus sit amet sem non, hendrerit venenatis magna. Nunc nulla leo, lacinia in sem ac, semper tincidunt nisl.

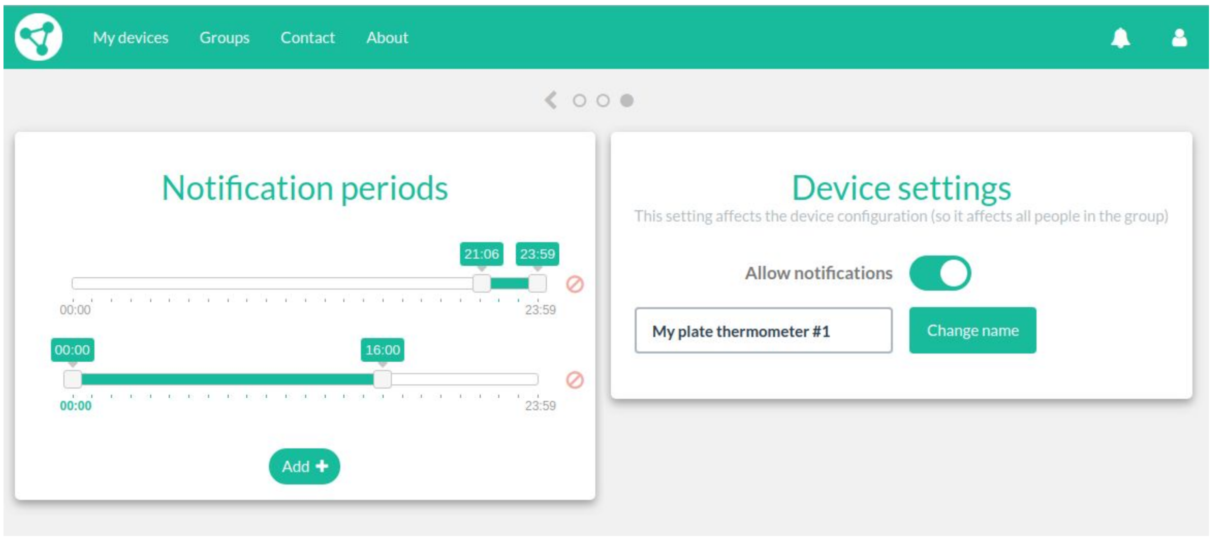
Main page of our application, you can easily access your devices and groups where you're in.



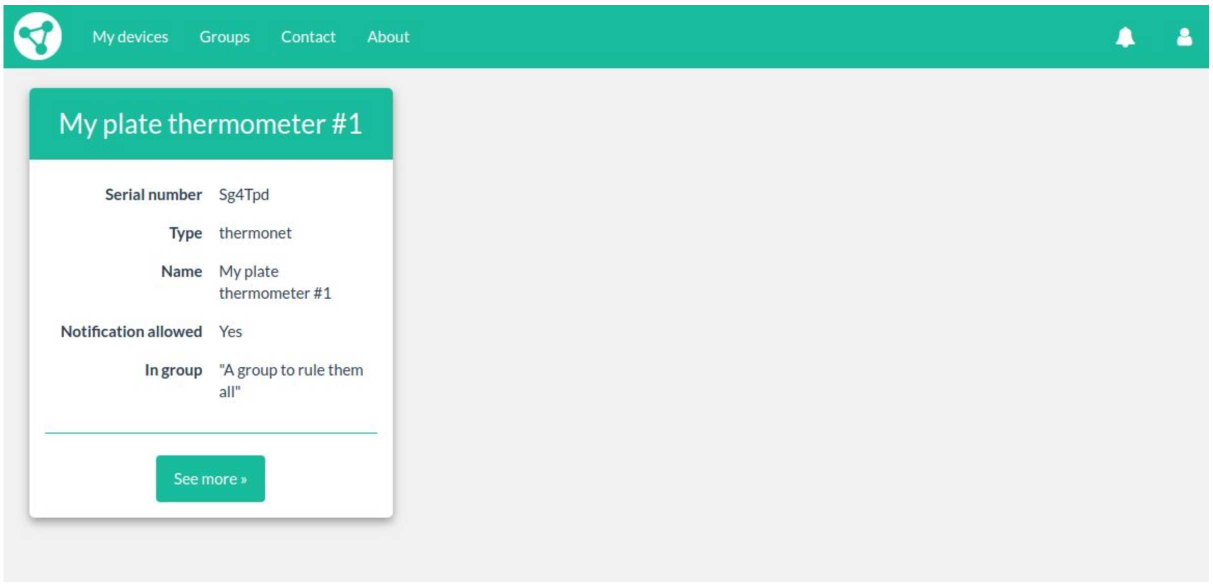
The sensor view of device consists of three tabs. The first tab combines indicators for device current status and time series graphs of ambient humidity and temperature sensor data.



The second tab of sensor view combines the infrared camera information. It has a live updating raw thermal camera view and a post-processed view that shows detected plates and their temperatures. A time series graph shows up the temperatures of detected plates over time.



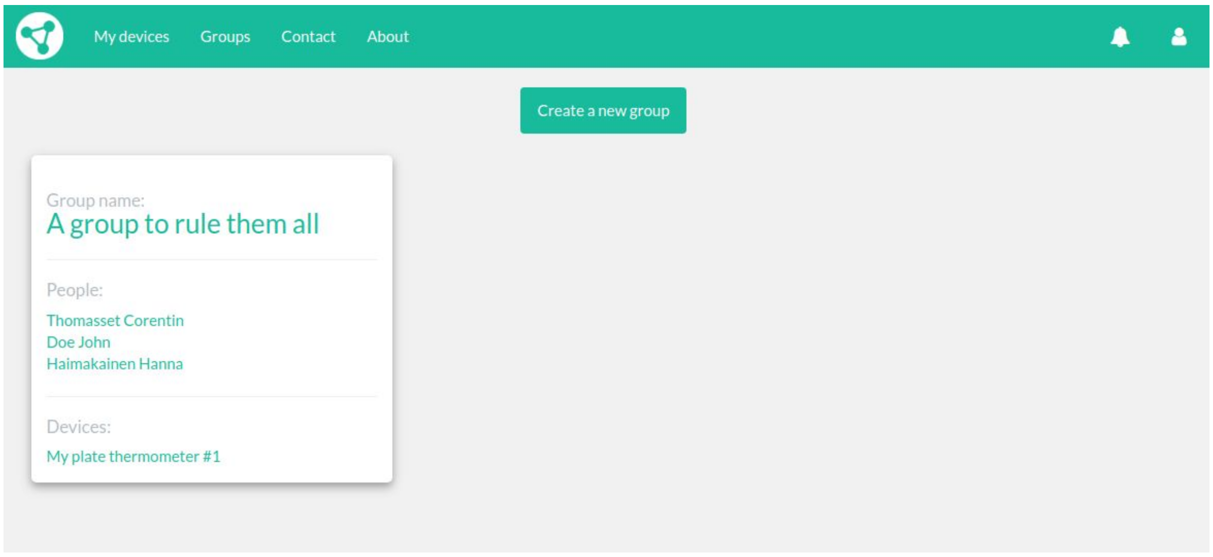
The configuration tab. This tab allow the user to change the name of the device, allow, or disallow notifications from this device and manage notification periods. They correspond to the time period where your plates should be turned off. If your plates are detected emitting heat during those intervals, you will receive warning notifications.



My plate thermometer #1

Serial number	Sg4Tpd
Type	thermonet
Name	My plate thermometer #1
Notification allowed	Yes
In group	"A group to rule them all"

The device view showing up the specifications and settings of devices connected to user account.

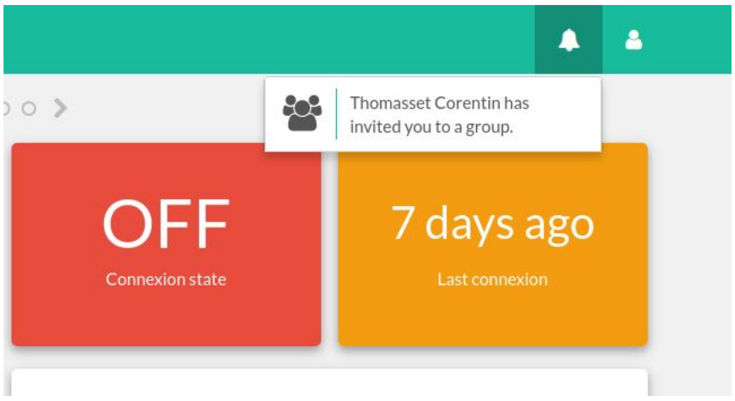


Group name:
A group to rule them all

People:
Thomasset Corentin
Doe John
Haimakainen Hanna

Devices:
My plate thermometer #1

The group overview, that allows the user to see in which groups they have subscribed and to manage subscribing information. Groups are useful if same stove is shared with multiple users.



Thomasset Corentin has invited you to a group.

Warnings and important messages are shown and addressed in the notification panel in the upper right edge of the page. Notification system supports multiple simultaneous notifications.

2) Database fields explanation

- **users**
    - **id**: unique number (permit to quickly identify a row of this table)
    - **firstname**: the firstname of the user
    - **lastname**: the lasname of the user
    - **email**: the email adress of the user
    - **password**: a hashed version of this user password (password not visible in the database)
    - **cookie_token**: a 60 characters long random string stored in cookie in the user browser so the user doesn't have to type it's credentials everytime.
    - **privileges**: a number defining the level of privileges of the user (1: read-only, 2: normal user, 3: admin)
    - **group_id**: a reference to the group in which the user is in (can be null if the user is in no group)
- **notifications**
    - **id**: unique number (permit to quickly identify a row of this table)
    - **user_id**: a reference to the user owning this notification
    - **seen**: a boolean value telling is the user has seen the notification or not
    - **text**: the text content of the notification
    - **link**: the link that will be triggered when the user click on the notification
    - **timestamp**: the timestamp at the creation of the notification
    - **option**: some option to personalize the notification (like change the icon)
- **devices_data**
    - **id**: unique number (permit to quickly identify a row of this table)
    - **device_id**: a reference to the device owning those data
    - **data**: the data in JSON format
- **groups**
    - **id**: unique number (permit to quickly identify a row of this table)
    - **manager_id**: a reference to user that has created this group. Only this user can manage the group
    - **name**: the name of the group
    - **invitation_token**: a 60 characters long random string that permits to identify this group for invitation purposes
- **devices**
    - **id**: unique number (permit to quickly identify a row of this table)

    - **name**: the name of the device
    - **serial_number**: an easily readable random 5 characters long string that should printed on the device so the user can add this device to it's account when he buy it
    - **token**: a 60 characters long random unique string that permits to identify the device when it send data to the server. So that unauthorized devices can't put stuff in the database.
    - **state**: a boolean depicting the state of the device (ON or OFF)
    - **last_reception_timestamp**: the timestamp of the last data reception
    - **allow_notification**: a boolean, to allow or disallow notifications
    - **notification_threshold**: the plate temperature threshold for notification periods. If a plate is above this threshold during a notification period, a notification saying "Your plates are on" will be send to the user.
    - **group_id**: a reference to the group in which the device is in (can be null if the device is in no group)
    - **owner_id**: a reference to the user that owns this device (the one that has registered it in his account)
- time_ranges
    - **id**: unique number (permit to quickly identify a row of this table)
    - **user_id**: a reference to the user that is linked to this time range.
    - **device_id**: a reference to the device that is linked to this time range.
    - **from**: the beginning of the time period.
    - **to**: the end of the time period.