



UKIYO

浮世絵

This doesn't look like a Smart Signage proposal

Brief Overview

Ukiyo is an encompassing solution to student oriented digital signage, giving both users and owners insights on information that is useful to them.

On the user side, tenants coming or going into the building are provided information on their day, following events, weather and all the smart mirror things.

Once a student gets closer, things change. We are able to identify them (with their permission, of course) and show them personalized information that is more relevant to them, such as their following classes.

However, to the building owners, Ukiyo is a strong data analysis platform, able to capture outlier communities and help them solve problems faster.

Is someone mining bitcoin in their dormitory? Is there a water leak? Are those more common in summer? These questions can be answered automatically by it.

The Device

The deployment of the service relies on the deployment of nearly 200 “smart signage devices”, which is fancy speak for what’s essentially a special use TV.

It is paired with a simple input device that makes the user able to respond to “yes/no” questions and scroll in case of need.

The display is powered by a single board computer (SBC, from now on) that would be similar to a Raspberry Pi. Sadly, the brand name comes with increased cost. More on this later!

Finally, there’s the optional webcam. The Device will use the camera to help perform various functions, but doesn’t technically need it.

In this document we’ll discuss how these are built, technical details, deployment costs and future expansion.

General Device Goals

1. Display a beautiful dashboard
2. Provide minimal interaction capabilities
3. Easy to use
4. Be cheap (relatively)

It makes sense to use off-the-shelf hardware considering the immense cost, zero flexibility and dubious privacy of pre-made Smart Signage solutions.

Due to this, the devices will be self-built. Like a good computer.

The Display

Modern LED backlit TVs are cheap and use little power. Depending on the screen size and resolution, these may come anywhere between 100 euro for a 720p 32" unit to 400 euro for a 50" 4K panel.

A middle ground for a 1080p 42", around **250** euro will be used for this project, although the SBC can support anything.

The Input Device

2 buttons, Yes and No, plus a *rotary encoder*: an oversized volume knob that will be used to scroll and select things, should the need happen.

The casing should be able to be 3D printed or fabricated using plastic extrusion for cheap. The switches for the buttons will be bought in bulk to a Chinese provider such as Outemu.

Wiring will directly connect to the SBC's general purpose control pins.

Cost of these control modules is essentially negligible, around 10 euro per unit.

3D Model

Duis autem vel eum iriure dolor in hendrerit in vulputate velit esse molestie consequat, vel illum dolore eu feugiat nulla facilisis at vero eros et accumsan.

SBC

The famous Raspberry Pi could be used, however it has either fallen out of the economic sense or the performance sense depending on what version we are talking about.

We support the use of a [Pine64 LTS unit](#) which will have many years of guaranteed replacements, has excellent upstream Linux support and can do anything that we need.

The complete functional kit is around 50 euro per unit plus power delivery.

Software

I. Alpine Linux

Embedded Linux distribution that has minimal attack surface and can be configured to run in read-only media, avoiding the usual issues with SBCs of dying SD cards.

This maximizes long term reliability, easily expanding into the 5 or even 10 year mark without maintenance of any kind.

Additionally, a single image will be configured and created, then copied to all SBCs.

II. Customized Firefox ESR

The SBC will boot into a fullscreen browser showing the main Ukiyo dashboard.

III. Motion

This software watches the camera unit and performs various tasks to help with the service's smooth operation.

Camera

A simple camera is connected to the SBC via its internal camera port.

The camera is feeding motion sensing software with information so we can detect whether there's someone possibly interested in the screen.

The camera introduces a 10 euro cost per unit and provides us with these features:

- **Power saving**

When no one is in front of the screen, it turns off. It can also turn back on when needed thanks to the HDMI CEC protocol. In case it isn't supported, something could be rigged up based on still image detection and sleep in modern TVs.

This reduces the long term environmental impact of a unit by more than half a megawatt hour over a 10 year period. (estimated via avoiding 3 hours of runtime a day) which avoids emitting around 700Kg of CO2!

[Source](#)

- **Privacy logoff**

The dreaded camera acts for the user instead of against the user, logging off any student that leaves the immediate vicinity of the TV.

Other factors will be used to un-personalize the screen (timeout, Wi-Fi proximity...)

With future advances in 5G technology or a more powerful SBC, processing could be offloaded to more powerful servers that are able to track faces and improve even more on this.

As discussed on the project submission, we considered the ethical, legal and security implications of such behavior and came to the conclusion that facial tracking is best left to movie dystopias.

Bill of Materials - BOM

Table of cost per unit:

TV	Example unit	220€
SBC and camera	Add a few to cart	60€
Control input	3D printed case	10€
	Random wires	1€
	2 Switches and a knob	5€
Total		~300€

Provided a total deployment of 200 units, the price reaches the 60,000€ in hardware without including installation costs and some degree of final R&D to prepare all the software required and iron out any quirks that may arise.

The final installation cost shouldn't exceed the 100k€ mark, even with massively upgraded displays or hardware.

As with most things, a small initial rollout is recommended.

The Software Side

We've developed a series of features that allow the display of this information in ways that are convenient and easy to digest to all members of the community.

Users

The product counts with several dashboards in charge of displaying different bits of information to the different types of users that can interact with it. So in order to understand all the features, we first need to describe the different people that will interact with Ukiyo

General Campus User

Each and every person in the campus will be able to interact with Ukiyo as an anonymous user. They'll be able to access different insights about the building they're at, the campus and pretty much any kind of information that might be of interest for anyone involved with the university.

Student

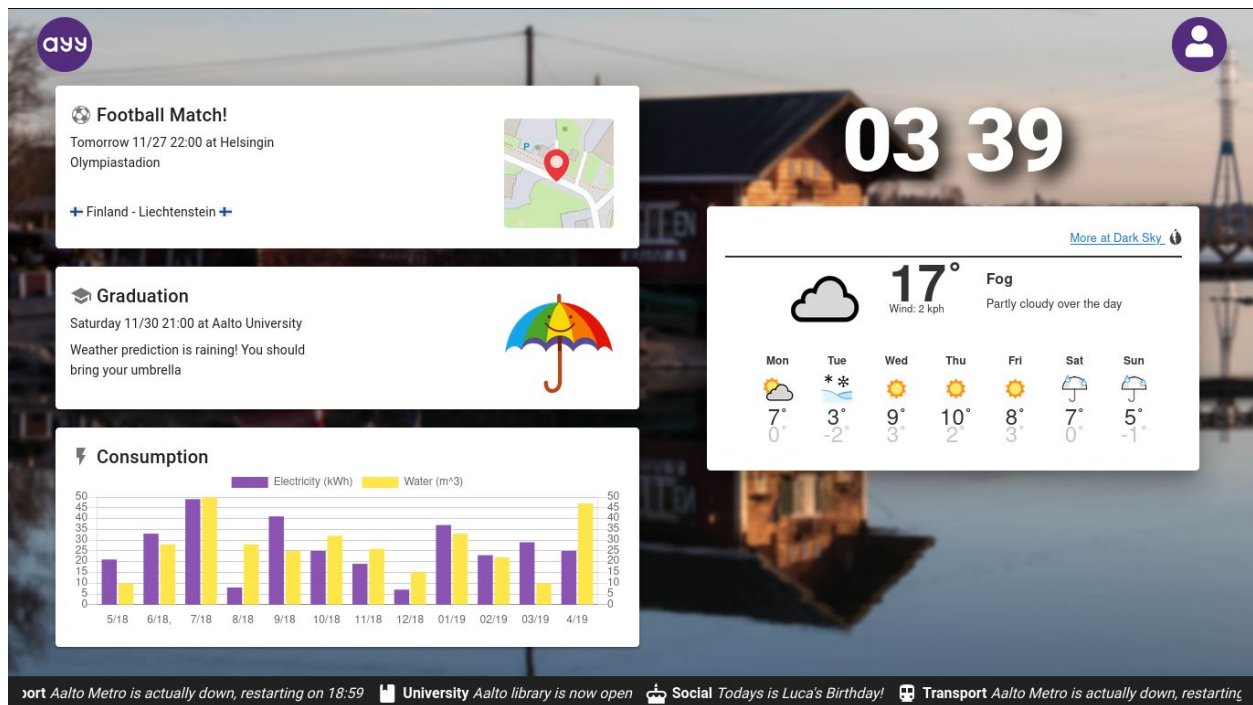
Students have a few more privileges since they are the ones most affected by anything that could affect the campus and its infrastructure and the ones that will benefit the most by the insights generated from the data collected by the university.

For this reason we've developed a **mobile app** that allows them to trigger exclusive features that will be described in detail later in this document.

Building Manager

Developing the product, we kept in mind that there are people in charge of using the data to ensure everything in the campus is up and running as well as improving as much as possible over time. This, as a result, motivated us to develop a **Web Interface** accessible to managers and members of the administrative side of the university that shows metrics more oriented towards the tracking of efficiency and development of the campus.

The Main Dashboard



The physical device that's installed in the different buildings of the campus will count with what we consider to be a "general dashboard" containing different pieces of useful information such as:

- The building's name/title
- Time
- Weather forecast
- Relevant events
- News Carousel

If a student "logs into" the Dashboard it will change to show them more relevant information in what we call *the personalization step*.

The Personalization Step

When a Student comes near one of the Devices, different methods will be employed to try to detect them and show their own personal Dashboard on the Device, instead of the standard general Dashboard aimed at “General Campus Users”.

How this works is highly technical and ever changing. Modern phone operating systems are getting a lot better at making it hard for others to track physically track them across buildings. The number of factors in this is enormous, but we’ll explain a few possibilities:

Bluetooth Beacons

The old concept for “intelligent places” was essentially to display a notification when the user got really close to a low power bluetooth pill.

The user would receive a notification to their phone, tap it, and the display would show their Dashboard.

However, support for these has been removed from modern Android and iOS because they were “annoying”.

On our side, it’s trivial to support them, so old phone users will have a better experience.

Geofencing

Here comes into play the app we developed. We can define a zone where your phone will do something when you enter it. This zone would be your building’s entrance.

Adding this data with others such as the camera presence and a background location request made possible by the geofence could provide enough *certainty of intention*.

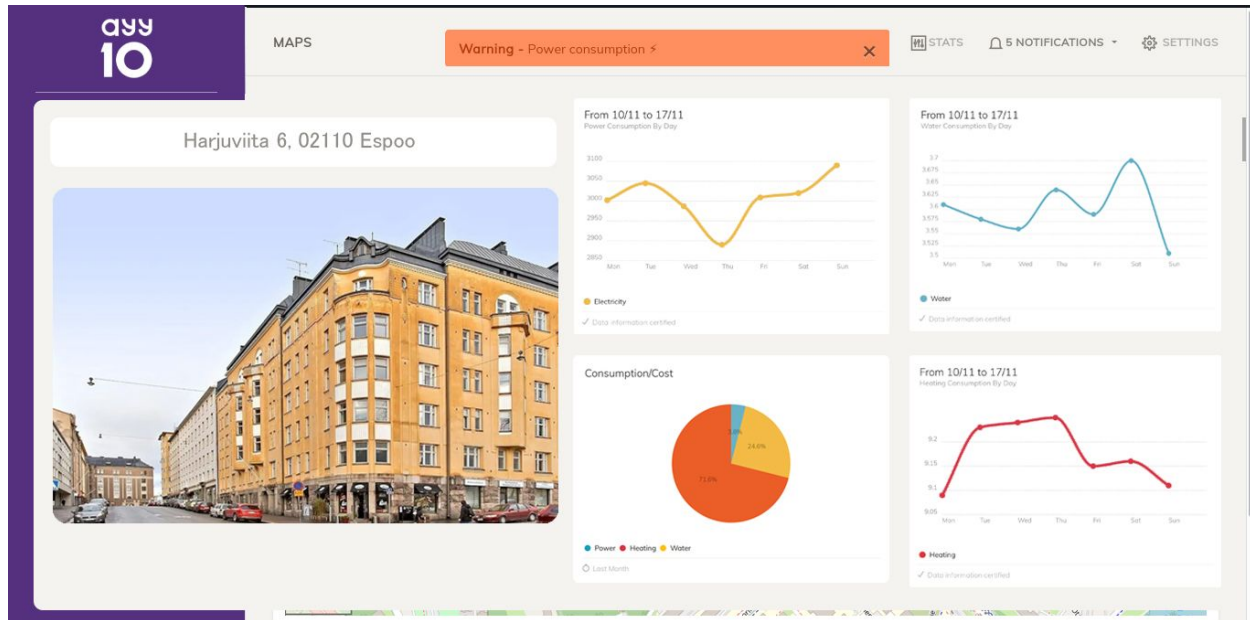
Wi-Fi RTT and beacon RSSI

Once we have determined that the user is near the Device, more precise measures can be taken to know if they are within a few meters of it. We are experienced in this from previous work during internships.

Manual polling

In case everything fails, the app is optimized for quick access to the manual connection tab. Were a Student wanting to quickly see something on the big screen, they would be only 3 taps away.

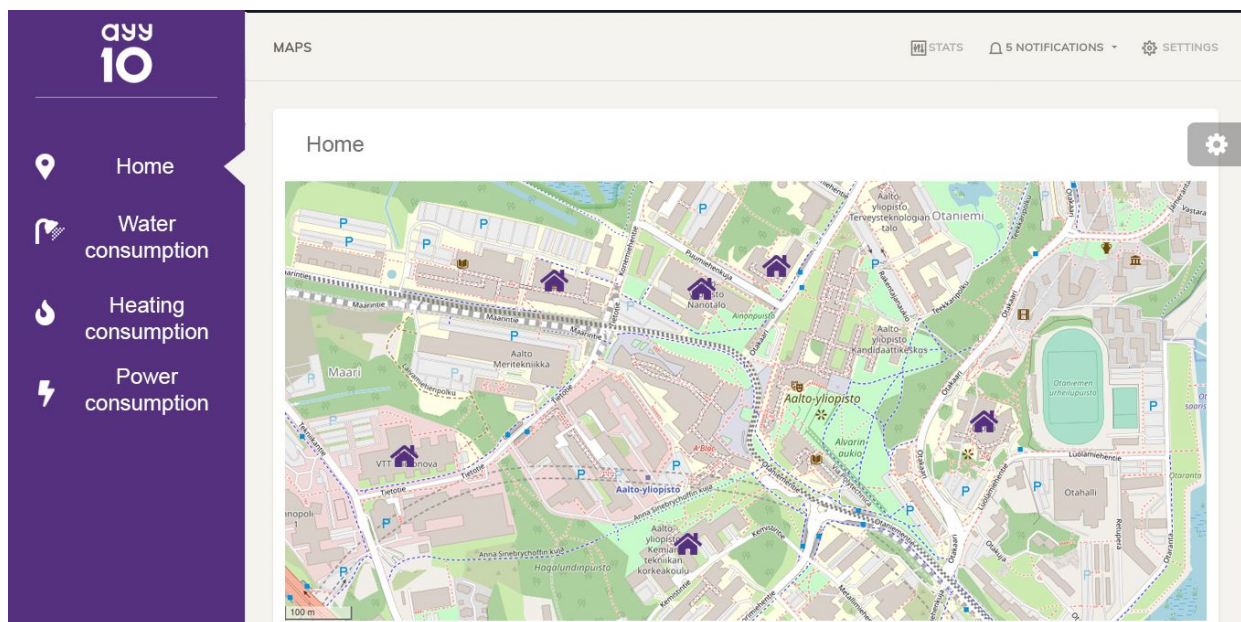
The Manager's Dashboard



This dashboard is designed to be accessed from a web browser by anyone in charge of overseeing the campus' development and efficiency (Building Manager type of user). It will be loaded with many different metrics and tools that aren't available to other types of users.

Aside from the charts and infographics, we've added Machine Learning (AI) models in a few of the features to allow managers to improve their planning process. So far, the app is capable of making accurate predictions of amounts and costs of the different services of a building such as electricity and water.

These Machine Learning models were trained using the datasets provided by the university for the Junction challenge, crossed with Business Finland's data about Helsinki and its surrounding areas, using their open API's and datasets which are also part of Business Finland's own challenge for the hackathon.



The App

Ideally, allowing the personalization to happen more easily would be a minor function of the app.

The functionality should be part of a bigger *Aalto University's Student Union* app that would provide more features outside of the scope of this project.


Gamified Planet Saving

However, a big goal for us is to improve sustainability and reduce the environmental impact of whatever we can touch.

With the data and capabilities available to us, we believe it is possible to create a system where the tenants themselves are invested in reducing their environmental impact.

A system of small rewards should be set up for communities displaying the lowest amounts of resource draw.

For example, communities in the top ten most power efficient could receive discounts in their rent (if applicable) or even small amounts of education credits.



This system would encourage people to be more mindful of their impact on the environment and even reduce maintenance costs as problems would be addressed more quickly.

Technical Implementation

The UI of the big screens, as well as the management webapp are developed using Vue.JS and communicates with several web services developed in .NET Core 3, using REST APIs and websockets.

One of these services was developed in .NET Core implementing Artificial Intelligence frameworks and tools, particularly Azure's Computer Vision cognitive service and it's Machine Learning platform. This service also communicates with Azure Cloud to leverage their power in ML and image recognition when needed, making it easier and clean to work with all the different data sets used in the solution.

Everything is dockerized and scales automatically if the necessity arises. Even with the pressure of being time limited in a hackathon it is possible to appreciate a certain level of clean code and planning in the services and it's code - we tried hard to follow SOLID, DDD CQRS and many other useful and interesting patterns.

The reference Android client uses the latest AndroidX and single-activity fragment manager design, it is also up to date to Android 10's very strict requirements on background and foreground Wi-Fi scanning.