# Introduction

There are few things in life worse than waking up to an annoying alarm clock, with the same repeating pattern every morning. It gets mundane, frustrating, and most likely unbearable after being startled several mornings consecutively.

# Project Description

How about waking up in a more elegant manner? With a smile on your lips in the morning? How is this achieved? This project will attempt to create an IoT product that solves the aforementioned issues. The project is done as part of the Internet of Things course at Aarhus University Herning.

The following challenges present themselves:

* How will the user be awakened in a pleasant manner?
* How will the user be awakened in time?
* What other ways can the morning routine be helped?

To make the solution a proper IoT product, and in line with the requirements of the IoT course at Aarhus University, the product will need to contain extra functionality that a non-IoT device would not. In addition to this, the original use cases must be kept intact, and a “graceful degradation” must be implemented, so that the system still offers value to the user. Even without an internet connection.

The proposed solution to these questions will come in the form of a little “radio alarm clock”-like device, that will play some music from the user’s library as the alarm (Either through streaming, or local library), and will fade in the tracks, to wake them pleasantly. This could perhaps be coupled with therapy lighting!

The Device has a cloud functionality, that pairs up with a Google Calendar, that has the user’s day-to day alarms gathered, and uses these to wake the user. In the case these services are not available, a preprogrammed alarm time can be relied upon.

The Device can help the morning routine of the user, by providing weather information, as well as news or other upcoming events from their calendar. I these services are not available, it will still be functional for every other purpose. It could even feature some Voice recognition software, but should be operable from buttons as well.

This project development will be loosely based around some of the activities in EUDP, which is the Procejt tool of choice at Aarhus University Herning [1].

# Requirements Analysis

I order to capture the most important requirements for the system-to-be, the EARS-requirement capture method, as suggested in EUDP, is used [2].

EARS includes the following classes of requirements: Ubiquitous, Event-Driven, State-driven, Option and Unwanted Behavior.

In addition to this, the requirements have been given a weighting; They either shall be fulfilled (Required for the system to be accepted), will be fulfilled (A future implementation is accepted), must be fulfilled (A strong wish, but no requirement), or be considered an improvement “to be” (A considered capability) [3].

For the system to be to be successful in fulfilling the criteria of the project description, the following requirements have been specified:

### Ubiquitous

* The system **shall** be capable of telling time accurately, and with minimal drift.
* The system **shall** be capable of connecting to the internet.
* The system **shall** wake up the user, at the specified times.
* The alarm **will** be pleasant to awaken to.
* Voice commands using online voice recognition is **to be** implemented.

### Event-driven

* When a specified time arrives, the system **shall** start the alarm.
* When the alarm starts playing, the system **will** slowly ramp up the sound.
* When the user has awoken, a weather message **will** be displayed.
  + When unavailable, the system **w**ill display a “weather unavailable” message.

### State-driven

* While connected to an online calendar, the system **shall** use the “wake-up” times specified therein.
* While the calendar is unavailable, the system **shall** use hardcoded “wake-up” times.
  + If possible, the system **must** store the latest calendar updates to use instead.
* While connected to an online playlist service, the system **must** stream songs from there as alarms.
* While the playlists are unavailable, on-device songs **shall** be played.

### Option

* Where possible, tough calculations **will** be performed off-device.
* Where user input is available, the input **must** be possible through button interface at least.
  + A microphone as interface is **to be** implemented if possible.

### Unwanted Behavior

* If system cannot access information while online, a debug message is **to be** implemented.

Furthermore, some of the requirements from the formal project description available from Aarhus University carry over as well, these consist of:

**Functional requirements**  
1  The device must be able to connect to the internet

1.1 Internet connection shall be via WIFI

  1.2.The device should preferably be able to connect  to AU’s “AU Gadget network”

2 Your device must be able to read data from a connected sensor, local to the device

2.1 a sensor can be anything that quantifies a physical measure, into an electrical signal, such as temperature, light, humidity, presence, movement, magnetism, pollution, etc.

3 Your device must be able to control an actuator

3.1 An actuator can be anything that translates an electrical signal into a physical quantity, such as, motors, servos, valves, heaters, displays, lamps, etc.

4 Your device must be capable of using data from a web service, to augment “what it does”, this could be weather data, traffic data, stock prices, twitter feeds, emails, rss-feeds or something different.

5 Your software and hardware design must be shared

5.1 You must create a public **github** account, and add relevant project files here

5.2 Hardware documentation, schematics, datasheets and pcb layouts are to be uploaded in pdf format

5.3 Software files are to be uploaded in raw source code format, e.g.  **.C, CPP, .h, .py**, etc.

**Technical requirements**

1 The technical platform can be a suited embedded platform of your choice, e.g. the Particle Photon, an ESP8266, a raspberry pi, beagle bone black or similar.

1.1. The platform shall have Wifi connectivity

1.2. The platform shall have available digital or analog I/O con connect sensors and actuators

Table 1 - Project requirements from Aarhus University Herning.

The project specific requirements, as well as the broader course-defined project requirements will both be considered and implemented. Seeing as the course-defined requirements naturally arise from the projects requirements anyhow, the implementation of one will likely entail the other.

# System Design

In broad strokes, the system will need to consist of the following hardware/software and services:

* A central processor, that does the simple logic – time etc.
* A media player, that will play the alarm media/songs when the time comes.
  + An amplifier to drive a speaker.
  + And the speaker itself.
* A calendar API to hold the “wake-up” times, and serve these to the logic.
* A weather API, to serve weather info to the logic.
* A display to display info, like the time, weather, etc.
* Some inputs to control the system, and request services.
  + Buttons to access manual controls in a menu.
  + Potentially a microphone to ask for services.
* Potentially a Voice recognition API to parse spoken controls.

The interfaces between these system components can be seen from Figure 1, where the connections are illustrated as simple inputs and outputs. The interactions between these interfaces will be explored more thoroughly, when specific possible solutions are found.

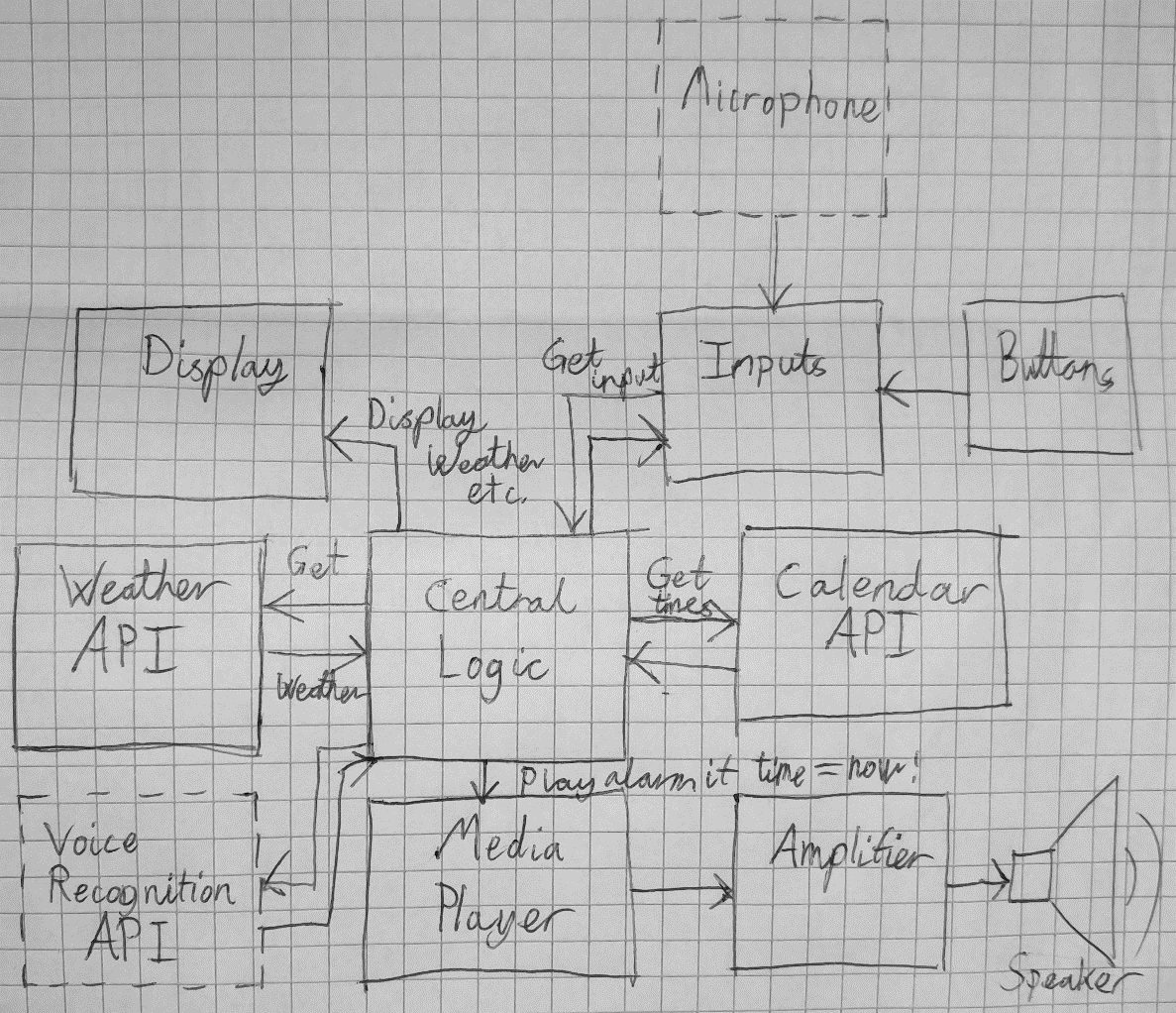


Figure 1 - A simple block diagram illustrating the construction of the system-to-be.

Among all these separate blocks, a lot of options are available to fulfill their individual requirements. For example, the central logic could be done in several ways depending on the platform:

* The logic could be written in C, and be a process running in a loop. Checking on webhooks online
  + This is a difficult approach, as C is a relatively low level language for this task.
* The logic could be run in a scripting language:
  + Bash scripts can handle logic, and is higher level than C.
  + Python Scripts run at a very high level, and have many external modules for several functions.
  + Node.js can run javascript on an embedded device.

## Hardware

One thing however, is abundantly clear. Since the device needs to access both WiFi, web API’s and play music, it would make the most sense to have an operating system on it. There are only a few platfoms available for this purpose, among others are:

* BeagleBone Black
  + Embedded Linux device, with WiFi, TCP/IP-stack, scripting and file system.
  + No directly accessible Audio, difficult to set up for this purpose.
* Raspberry Pi 3
  + Embedded Linux Device with WiFi, TCP/IP-stack, scripting and file system.
  + Includes multimedia drivers, and has several community supplied overlays for audio, along with a TRS output.
* Raspberry Pi Zero W
  + Embedded Linux device with WiFi, TCP/IP-Stack, scripting and file system.
  + Includes multimedia drivers, and has several community supplied overlays for audio, along with a TRS output.
  + Cheap, has small form factor.

Few other candidates apply for a project of this shape. As such, it seems most logical to pick the cheapest and most capable of the devices. From these three listed platforms, Raspberry Pi Zero will be the platform of choice.

## Software

Et par smarte libs til at styre dato i Python:

Calendar functions in python:

<https://developers.google.com/google-apps/calendar/quickstart/python>

<https://docs.python.org/2/library/datetime.html>

Spotify playback in Python:

<https://pyspotify.mopidy.com/en/latest/api/sink/#spotify.AlsaSink>

Spotify playback in JS:

<https://developer.spotify.com/web-api/>

Google assistant API:

<https://developers.google.com/assistant/sdk/develop/python/>

<https://developers.google.com/assistant/sdk/develop/grpc/integrate>

pyAudio:

<http://people.csail.mit.edu/hubert/pyaudio/>