TÉCNICO LISBOA

Sensing and Visualizing the Deep Blue Ocean Project

Pedro João Lourenço Ribeiro

Thesis to obtain the Master of Science Degree in

Information Systems and Computer Engineering

Supervisor: Prof. Duarte Nuno Jardim Nunes

Examination Committee

Chairperson: Prof. Alberto Manuel Rodrigues da Silva
Supervisor: Prof. Duarte Nuno Jardim Nunes
Member of the Committee: Prof. Manuel Fernando Cabido Peres Lopes

November 2018

Dedicated to my Parents and siblings...

Acknowledgments

First of all, i would like to thank my advisor, professor Nuno Nunes for all the support, availability, and knowledge transmitted.

I also would like to thank to Dinarte Vasconcelos, Miguel Ribeiro and Marko Radeta for the all help and contribution, with both comments and suggestions that allowed me to overcome some obstacles.

A big thank you to my family and in special to my uncle that help me to build the case that was fundamental for the conclusion of the project.

Last but not least i want to thank to my friends for all the support that helped me and motivated me throughout this project.

Resumo

O generalizar da disponibilidade da tecnologia informática tem vindo a criar oportunidades para intervenções com o intuito de promover a sustentabilidade ambiental e a consciência ecológica por parte de novos utilizadores ou existentes. Estes são domínios onde os contextos políticos e culturais têm implicações mais amplas, que estão apenas a começar a ser exploradas pela comunidade científica. Este projeto visa a aplicação de tecnologias de detecção de baixo custo para a zona mesopelágica , utilizando os pescadores e os seus aparelhos como veículos para a implantação de novos sensores acústicos e de imagem que nos permitem explorar o fundo do Oceano. O projeto envolverá a adaptação de plataformas existentes (como soundtrap.io e mataki.org) para implantação da pesquisa no Oceano Atlântico. O projeto envolve a implantação e teste do sensor em condições reais em colaboração com o Museu de História Natural do Funchal, na Ilha da Madeira. O projeto usará kits de desenvolvimento de hardware de baixo custo (https://store.particle.io/) para prototipagem.

Palavras-chave: Biologging, Tags Eletrônicas, GPS, Aplicação para dispositivos móveis, Servidor

Abstract

The widespread availability of computing technology is creating opportunities for interventions to promote environmental sustainability and ecological consciousness on the part of existing or potential technology users and new ones. These are domains where the political and cultural contexts have wider implications, which are just starting to be explored by the scientific community. This project aims at applying low-cost sensing technologies to the oceanic Twilight Zone using fishermen and their apparatus as vehicles for deploying new acoustic and image sensors for exploring the deep Ocean. The project will involve adapting existing platforms (such as soundtrap.io and mataki.org) for effective research deployment in the Atlantic Ocean. The project involves deploying and testing the sensor in real-world conditions in collaboration with the Natural History Museum of Funchal in Madeira Islands. The project will use low-cost hardware development kits (https://store.particle.io/) for prototyping.

Keywords: Biologging, Electronic Tags, IoT, Mobile Phone Application, Server

Contents

List of Tables

List of Figures

Chapter 1

Introduction

In an age marked by the conquest of the space we still know so little about our oceans. They are the lifeblood of planet Earth and humankind. They hold 97% of the planet's water and they occupy almost three-quarters of the planet. They support the greatest biodiversity of the planet and they are also one of the largest carbon reservoirs holding 54 times more carbon than the atmosphere. Exploring the ocean is a really big challenge because the technologies that currently exists are really expensive, which limits the amount of work that can be done. On the other hand, with the emergence of the climate changes, it has become increasingly necessary to explore and understand the impact that pollution will have on the lifestyle of the marine animals and the ocean itself. As we know we are moving towards an ubiquitous world where the technology will blend in with the environment and become almost invisible. In recent years, we have seen an exponential increase in the number of small computers, that made possible new discoveries in a variety of areas due to their relative low cost, small size and adaptability to almost every circumstance. As Mark Weiser, principal scientist and manager of the Computer Science laboratory at Xerox Parc, wrote in a magazine.[1]

> *"The goal is to achieve the most effective kind of technology that which is essentially invisible to the user. To bring computers to this point while retaining their power will require radically new kinds of computers of all sizes and shapes to be available to each person. I call this future world "Ubiquitous Computing"."*

On the other hand, in recent years the tourism in Portugal has gained greater importance in the economy of the country with millions of people coming here to see the natural beauty that it has to offer. With more people coming, new business appear, like the tours to see whales and dolphins in Madeira, this can affect the species that live in our oceans which is why monitoring them is so important to prevent future disasters. In order to efficiently monitor the species behavior, Biologists need new approaches and equipment's. But the data that is gathered by the monitoring can also be useful for companies to better serve tourists and at the same time raise awareness of the risks that marine life faces.

Nowadays we are living in a trend where "the growing availability of computing technologies such as mobile GPS enabled devices, capable of image capturing and processing, ecology and environmental science are now capitalizing on the "talents and geographical spread of non-specialists 'citizens', with

spare time, curiosity and a smart phone". [2, 3] This is allowing members of the general public, typically non scientists, to generate a new type of data collection and analysis that can be called Citizen Science. The amount and diversity of current citizen science projects is astonishing. We have volunteers monitoring the night sky for light pollution, different type of birds, free divers capturing the extent of oceanic pollution etc. With this new citizen science projects we can expect an exponential growth on the data available which in the long term can help us to better understand what surround us.

1.1 Objective

The main goal of this thesis is to gather information about marine mammals and the ocean, by using inexpensive and small micro-controllers and then present this information to both biologists and users.

1.2 Proposed Approach

In order to achieve the main goal, our approach will be to create software in a modular way so that everyone can use and adapt our technology for future investigations. Furthermore we will be able to connect the data gathering devices with mobile devices to improve the spread and availability of the data to everyone. This project will be comprised of the following steps:

- The initial investigation about Biologging and the techniques that are used today;
- Analyses and extraction of the requirements needed for the project;
- Initial build of the device model;
- Build the necessary infrastructure for communication;
- Develop of the application for mobile devices;
- Testing and documenting and requirements check;
- Deployment of the project and monitoring the results;
- Maintenance and enhancement of the project with new features and feedback from its users.

1.3 Thesis Outline

This thesis is organized as follows: Chapter 2 We will talk about work that as already been done in this area as well as introduce some new concepts like Biologging. In Chapter 3 we will describe our solution, telling what we have done and how we did it. In the Chapter 4 we will talk about the tests performed and their respective results. Finally, we will draw a conclusion to this project in the Chapter 5.

Chapter 2

Related Work

2.1 Citizen Science

2.2 Citizen Science and Biologging

In introduction we talked a little about a new trend called citizen science. Citizen science is a movement that begun recently with the increasing availability of cheap devices, such as microcontrollers, and the fact that the computational power that we carry in our pockets today is absurdly larger than what we had a few years ago. The benefits are assumed to extend beyond the production of large databases. Many people argue that participants will increase their understanding about the process of science through the engagement in authentic science.[2, 4]

There are currently many research programs which monitor and identify bats, bees, birds, flies and slugs for example, but citizen scientists don't need to work outdoors. Non-specialists have been trained to scan satellite imagery for wildebeests in the Serengeti, penguins in the Antarctic, and African migrant groups affected by environmental catastrophes who require emergency aid.[3]

As we know studding wild animals can present major challenges, so in order overcome them biologging technology started to appear. Biologging can be described as "the use of miniaturized animal-attached tags for logging and/or relaying of data about an animal's movements, behavior, physiology and/or environment. Biologging technology substantially extends our abilities to observe, and take measurements from, free-ranging, undisturbed subjects, providing much scope for advancing both basic and applied biological research."[5]

In the last 20 years we have seen a rapid development in technology and miniaturization which made possible that biologging could be applied to almost every animal even the ones that live in harsh environments for humans.[6] This has led to particularly rapid advancements in the marine realm where direct observations are almost impossible.

Electronic tagging of marine animals is increasingly being used by scientist to track their movements. These tags allow us to see where and how marine animals travel, and then we can correlate with the ocean environment and understand what drives their movements.[7, 8]

2.2.1 Electronic tagging of marine animals

In this subsection we will show different types of Electronic tags of marine animals that used biologging techniques for gathering data about their day to day life.

Archival tags

Archival tags are small data loggers fig. 2.1 that can record swimming depth, internal and external temperatures and ambient light levels. They can record the data for up to 10 years depending on the battery and the tag's sampling frequency. Their biggest disadvantage is that they need to be retrieved in order to download the data so they can only be used in species that have a high likelihood of being recaptured.[7, 8]

One example of a study [9] conducted using archival tags was placing these tags in Bigeye tuna in the Equatorial Pacific. They managed to recover 74 of the 451 tags that where deployed in large part because of the reward that they offer to those who found them.

Figure 2.1: Archival tag model LTD2310 manufactured by Lotek Wireless

Pop-up satellite archival tags

Pop-up archival transmitting (PAT) tags fig. 2.2 are tags that are placed externally in marine animals. Once they reach a preset time they detach and rise to the surface and start sending the data to the Advanced Research and Global Observation Satellite (ARGOS) satellite network. [7, 8, 10] Scientists can then collect and analyze the data gathered. This system is one of the most used because it does not have to be physically retrieved like an archival tag for the data to be available making it a viable, fishery independent tool for animal behavior and migration studies.

Figure 2.2: Pop-up satellite tag

Satellite positioning tags

Satellite positioning tags fig. 2.3 are also attached externally to the animals, but since they need to send information regularly to ARGOS [10] or Global Positioning System (GPS), they can only be attached to animals that spend enough time on the surface so that the antenna can be outside of water. For this reason they are mostly used in animals such as marine mammals, sea turtles, sea birds and some sharks. [7, 8]

Figure 2.3: Satellite positioning tag fitted in a juvenile white shark

GSM tags and Acoustic tags

Global System for Mobile communications (GSM) tags are like the Satellite positioning tags but communicate using GSM network instead of using a satellite communication. Because of this they can only be used when animals are in coastal waters where the signal reaches. Although Acoustic tags are in large expansion they can not be used in this project because they emit sounds that would affect the whales

behavior. So they are not relevant in the current topic.

Digital acoustic recording tags

Digital acoustic recording tags fig. 2.4 are the ones that are closer to our project idea. They utilize a suction cup to attach to the animal. Everything else is like the PAT tags. These tags are being used to characterize whale movements and sub-surface behavior, including the kinds of vocalizations used while diving and foraging, and their responses to human activities.[7, 8, 11]

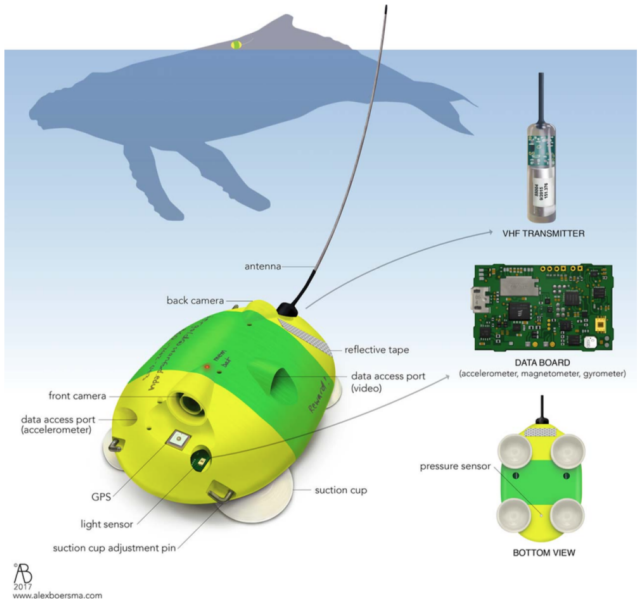

Figure 2.4: Scheme of a Digital tag.[11]

2.2.2 Low cost Biologging tags

With the growth of emerging markets large technology companies and foundations started to develop computers to the masses. This allowed the appearance of affordable microprocessors that can be seen almost everywhere today. The best example is the raspberry pi [12] that was first thought to motivate children to develop something new. In 2012 the first raspberry was launched capturing immediately the attention of millions of people. This type of products are really well supported by the open source communities which have enabled the development of new projects with substantially lower costs.

Raspiwhale [13] is a project similar to the one we are going to develop that utilizes a low cost microprocessor equipped to dive to great depths in the sea aboard a whale.

Figure 2.5: Model of a Raspiwhale.

2.3 Microcontrollers

A microcontroller is a small computer on a single integrated circuit. It has a processor, memory for storage, Random Access Memory (RAM) to run the programs and for data input and output interfaces. The main difference between a microcontroller and a microprocessor is that the first one has all the structure and components necessary for computing in a single chip, while the other uses external memory for program and data storage.

Microcontrollers are designed to perform specific tasks. For example, keyboards, mouse, washing machine, pen drive, etc. Since the applications are very specific, they need small resources like RAM, ROM, I/O ports etc and then they can be embedded on a single chip, which leads to reduced sizes and costs. With the development of this kind of chips nowadays we can find a lot of this microcontrollers with everything that we need like Wi-Fi, GPS, GSM, etc.[14]

2.3.1 Particle Photon

Particle (formerly Spark) is a small Kickstarted company being the one responsible for one of the first Internet of Things (IoT) development boards to hit the market: the Spark Core. However, at 33 euros the Core turned out to be too expensive to cater to most makers, so the people at Particle revised their

design and released their second board, the Photon, fig. 2.6. The Photon costs 16 euros, which is inexpensive relative to its predecessor. It runs on 32-bit ARM Cortex M3 Architecture at 120Mhz and has a full suite of I/O pins. The only areas where the Photon skimps are power consumption and flash memory, with only 1MB of flash available to the user.[14, 15]

Features:

- Particle PØ Wi-Fi module
  - Broadcom BCM43362 Wi-Fi chip
  - 802.11b/g/n Wi-Fi
  - STM32F205RGY6 120Mhz ARM Cortex M3
  - 1MB flash, 128KB RAM
- On-board RGB status LED (ext. drive provided)
- 18 Mixed-signal GPIO and advanced peripherals
- Open source design
- Real-time operating system (FreeRTOS)
- Soft AP setup
- FCC, CE and IC certified

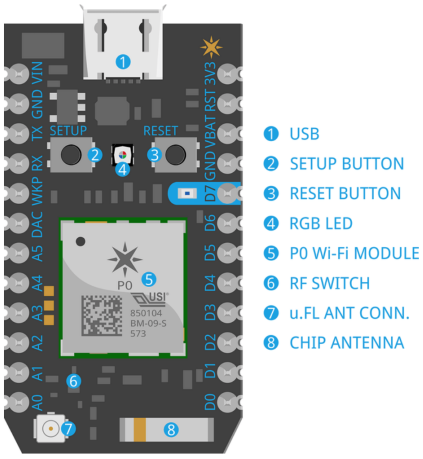

Figure 2.6: Pin markings for Photon.[15, 16]

Memory Map STM32F205RGY6 FLASH LAYOUT OVERVIEW:

- Bootloader (16 KB)
- DCT1 (16 KB), stores Wi-Fi credentials, keys, mfg info, system flags, etc..
- DCT2 (16 KB), swap area for DCT1
- EEPROM emulation bank 1 (16 KB)
- EEPROM emulation bank 2 (64 KB)
- System firmware (512 KB) [256 KB Wi-Fi/comms + 256 KB hal/platform/services]
- Factory backup, OTA backup and user application (384 KB) [3 x 128 KB]

2.3.2 Arduino Uno

Arduino is an open-source electronics platform based on easy-to-use hardware and software.[17] Arduino boards are able to read inputs - light on a sensor, a finger on a button, etc - and turn it into an output - activating a motor, turning on an LED, publishing something online. You can tell your board what to do by sending a set of instructions to the microcontroller on the board. This boards uses the Arduino programming language (based on Wiring), and the Arduino Software (IDE), based on Processing.[15, 18] In the fig. 2.7 we can see the Arduino Uno module.

Arduino was born at the Ivrea Interaction Design Institute as an easy tool for fast prototyping, aimed at students without a background in electronics and programming. As soon as it reached a wider community, the Arduino board started changing to adapt to new needs and challenges, differentiating its offer from simple 8-bit boards to products for IoT applications, wearable, 3D printing, and embedded environments. All Arduino boards are completely open-source, empowering users to build them independently and eventually adapt them to their particular needs. The software, too, is open-source, and it is growing through the contributions of users worldwide.[15, 17]

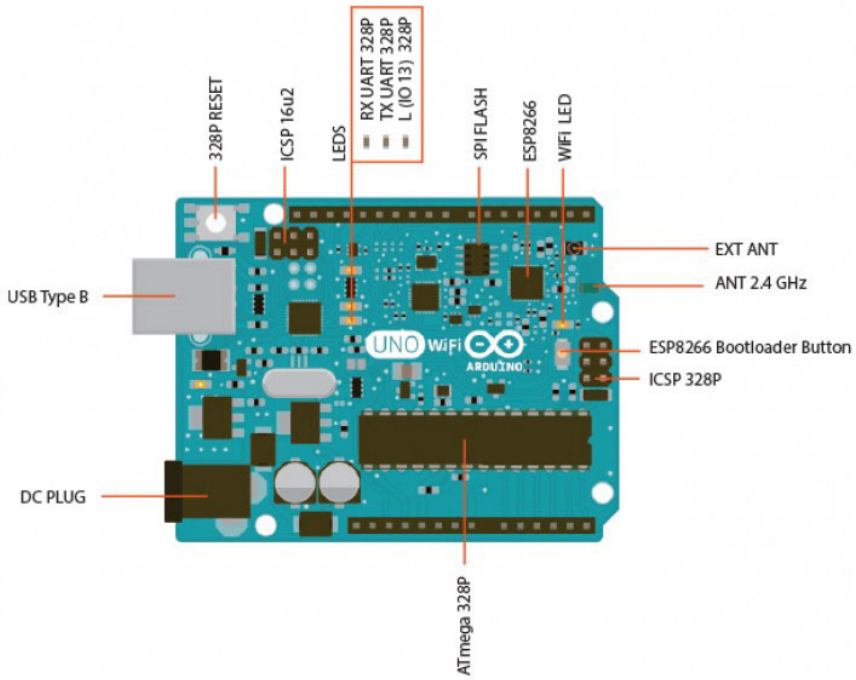

Figure 2.7: Diagram of Arduino Uno

Advantages of Arduino:

- Inexpensive (Arduino modules cost less than 42 euros);
- Cross-platform (The Arduino Software (IDE) runs on Windows, Macintosh OSX, and Linux operating systems);
- Simple, clear programming environment;
- Open source and extensible software;
- Open source and extensible hardware.

The main features are:[15, 19]

- 32 flash memory, 2 of SRAM and 1 of EEPROM (library is required to read or write into this);
- Clock speed is 16 ;
- USB connectivity (data communication between PC and Arduino);
- Powered via the USB connection or with an external power supply (6 to 20), works on 5 ;
- Supports I2C and SPI communication (libraries);
- 10-bit Digital-to-Analog (D/A) output (0-1023).

2.3.3 Adafruit Feather HUZZAH

Feather is the new development board from Adafruit, fig. 2.8. At the Feather HUZZAH's heart is an ESP8266 Wi-Fi microcontroller clocked at 80 and at 3.3 logic. This microcontroller contains a Tensilica chip core as well as a full Wi-Fi stack. We can program the microcontroller using the Arduino IDE for an easy-to-run Internet of Things core. Has connected a high-quality SiLabs CP2104 USB-Serial chip that can upload code at blistering 921600 baud for fast development time. It also has auto-reset so no noodling with pins and reset button pressings. The CP2104 has better driver support than the CH340 and can do very high speeds without stability issues [15, 20].

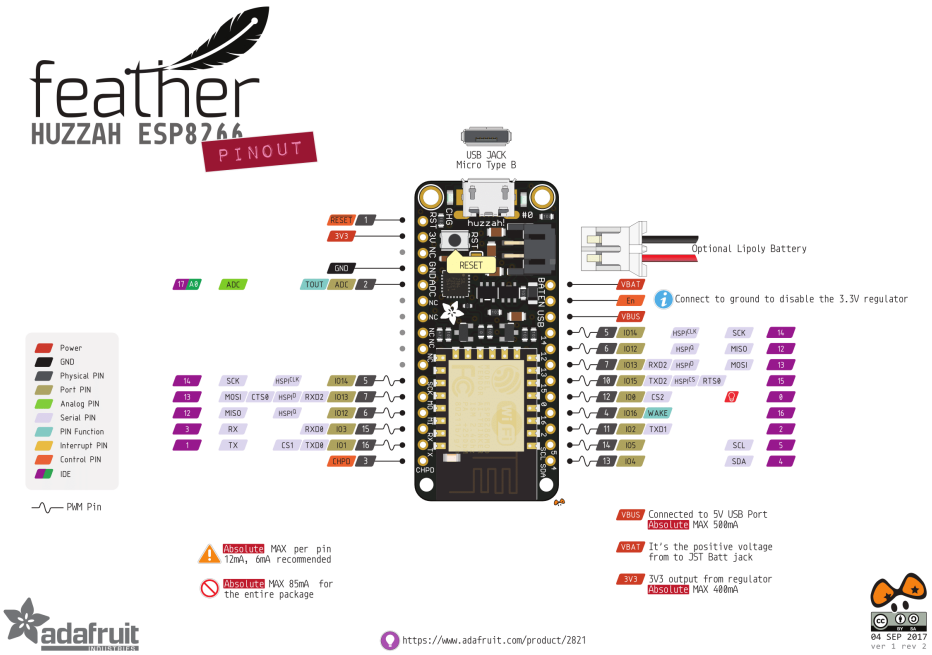

Figure 2.8: Adafruit Feather HUZZAH ESP8266.[15, 20]

In order to make it easy to use for portable projects, Adafruit added a connector for any of theirs 3.7V Lithium polymer batteries and built in battery charging. You don't need a battery, it will run just fine straight from the micro USB connector. But, if you do have a battery, you can take it on the go, then plug in the USB to recharge. The Feather will automatically switch over to USB power when its available.[20]

Features:

- Measures 2.0" x 0.9" x 0.28" (51 23 8) without headers soldered in;
- ESP8266 @ 80 with 3.3 logic/power;
- 4 of FLASH (32);
- Built in Wi-Fi 802.11 b/g/n;
- 3.3 regulator with 500 mA peak current output;
- CP2104 USB-Serial converter onboard with 921600 max baud rate for speedy uploading;

- Auto-reset support for getting into bootloader mode before firmware upload;
- 9 x GPIO pins - can also be used as I2C and SPI;
- 1 x analog inputs 1.0 max;
- Built in 100 LiPoly charger with charging status indicator LED, can also cut a trace to disable the charger;
- Pin 0 red LED for general purpose blinking. Pin 2 blue LED for boot loading debug and general-purpose blinking;
- Power/enable pin;
- 4 mounting holes;
- Reset button.

2.3.4 Comparison of the microcontrollers considered

In the Table 2.1 is possible to observe the comparison between the three proposed microcontrollers. Since we are aiming at a modular device with the cheapest price possible all this microcontrollers fulfill this requirement. The Adafruit Huzzah is the one that has the most flash memory and built-in battery, yet is the one with the smallest amount of digital and analog pins. Comparing now the Particle Photon with the Arduino Uno, we can see that both of them have an equivalent number of pins, however the Photon has much more memory, a greater clock speed and an antenna. The only disadvantage is that it does not come with the ability to connect the battery directly to the microcontroller, requiring a regulator.

Table 2.1: - Comparison between the principals features of the three boards considered in this project.

	Particle Photon	ADAFRUIT FEATHER HUZZAH	ARDUINO UNO R3 (ATMEGA328)
CLOCK SPEED	120MHz	80MHz	16MHz
FLASH MEMORY	1MB	4MB	32KB
DIGITAL PINS	18	9	14
ANALOG PINS	6	1	8
ANTENNA TYPE	PCB and uFl	PCB	No
BATTERY CAPABILITY	No	Yes	No
ONLINE SERVICE	Yes	Yes	No
Cost(USD)	$19	$16	$25

One of the biggest advantages in comparison with other products is the connection between the photon and the Particle web service. New programs can be pushed to boards over the internet and the service can connect to If This, Then That (IFTTT). [21, 22]

2.4 Communication

In recent years the Low-Power Wide-Area Network (LPWAN) represent a new trend in the evolution of communication designed to enable broad range of IoT applications. The main difference between the currently existing communication technologies(e.g. 4G, 5G) and LPWAN is that the first ones focus on the high data rate for each device, while in the second one, data rates are intentionally kept low and are traded for long communication ranges. Another feature of this system is the energy efficiency, since many of the end devices are expected to be powered by a battery.[23]

With these characteristics it's obvious that we can't use it to some data hungry applications like streaming. In other hand LPWAN suit quite well the requirements of smart metering applications to monitor the environment for example. Besides they can be employed for tracking devices and monitoring people's well-being.

2.4.1 Sigfox

Sigfox [24] is a French multinational responsible for developing wireless networks that connect very low-power devices, such as electricity, water and gas meters, home appliances or amateur weather platforms, to the Internet or, in special cases, to each other. As stated early, the networks created use the LPWAN.

The SigFox business model takes a top-down approach. The company owns all of its technology from the back-end data and cloud server to the endpoints software. But the differentiating factor is that SigFox is essentially an open market for the endpoints. They give away its endpoint technology to whatever silicon manufacturer or vendor wants it as long as certain business terms are agreed upon. Large manufacturers like STMicroelectronics, Atmel, and Texas Instruments make SigFox radios. They follow this approach because they think that allowing the application to be really inexpensive is the way to drive people to its market.[25]

SigFox uses a ultra narrow band (UNB) to establish a 2-way communication service trough a global radio network between equipment's and a proprietary base-station. As soon as the base-station receives a message from one of the devices in the area, it sends the message to the SigFox cloud which will then make it visible to the user own cloud, where he can see and manipulate the information received.[26, 27]

As we can see in the fig. 2.9 the messages need to be small (12 bytes Uplink, 8 bytes Downlink) and each device can only sent up to 140 messages a day to the cloud. on the other hand UNB technology makes it possible to send messages on channels with a bandwidth of less than 200 Hz. The range of this type of signal can exceed 30 km in rural areas. Another of the advantages of this technology is the strong immunity to noise: as the signal has a very small bandwidth, the receivers will have filters also

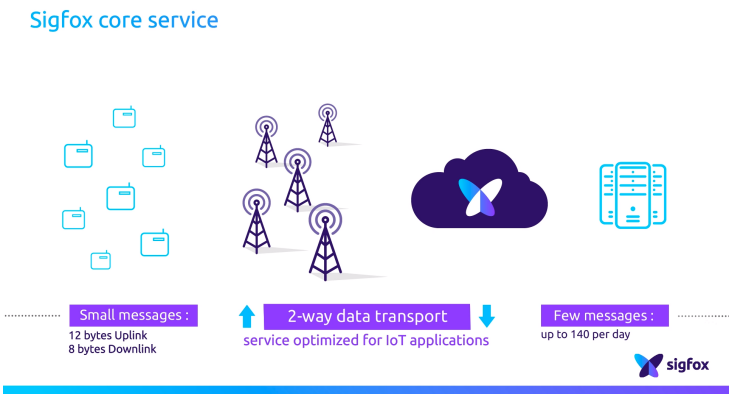

Figure 2.9: Illustration of the operation of a SigFox network.[26, 27]

with a small bandwidth, thereby removing a large part of the signal noise.

2.4.2 LoRa Alliance

LoRa Alliance is a open and non-profit association of members that understands that the IoT era has arrived. Their mission is to standardized low power wide area networks to enable the IoT. The LoRa Alliance also offers certification programs to ensure compliance and the ability to commercialize your products around the world.[28]

LoRa Wide Area Network (LoRaWAN) is a LPWAN specification intended for wireless battery operated things in a regional, national or global network. LoRaWAN targets key requirements of IoT such as secure bi-directional communication, mobility and localization services. The LoRaWAN specification provides seamless interoperability among smart things without the need of complex local installations and gives back the freedom to the user, developer, businesses enabling the roll out of IoT.[29]

The main features of LoRa are:

- Easy to install
- Extremely economical
- Flexible to adapt
- Scalable
- Bi-directional
- Secure and encrypted

LoRaWAN has several different classes of end-point devices to address the different needs reflected in the wide range of applications:[29]

- *Bi-directional end-devices (Class A):* End-devices of Class A allow for bi-directional communications whereby each end-device's uplink transmission is followed by two short downlink receive windows. This Class A operation is the lowest power end-device system for applications that only

require downlink communication from the server shortly after the end-device has sent an uplink transmission. Downlink communications from the server at any other time will have to wait until the next scheduled uplink.

- *Bi-directional end-devices with scheduled receive slots (Class B):* Class B extends Class A by opening extra receive windows at scheduled times. In order for the End-device to open its receive window at the scheduled time it receives a time synchronized Beacon from the gateway. This allows the server to know when the end-device is listening.
- *Bi-directional end-devices with maximal receive slots (Class C):* End-devices of Class C are always able to receive data from the gateway, except only when they are transmitting.

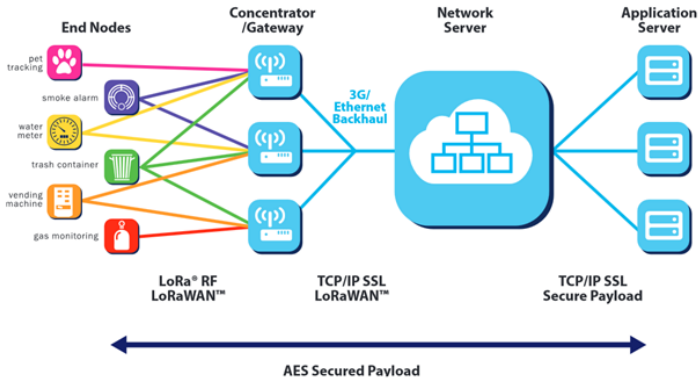

Figure 2.10: LoRa network illustration.

2.4.3 Weightless special interest group

Weightless is a set of LPWAN open wireless technology standards for exchanging data between a base station and thousands of machines around it. Weightless SIG (special interest group) was founded in 2008 with the mission of standardizing LPWAN. Weightless is the only truly open-standard that operates in sub-1 GHz unlicensed spectrum. There are three versions of Weightless that serve different purposes:[30, 31]

Weightless-W: This Weightless-W open standard is designed to operate in TV white space (TVWS) spectrum. Using TVWS is attractive in theory, because it takes advantage of good ultra-high frequency (UHF) spectrum that's not otherwise in use. However, in practice, it can be quite difficult because in one city you might have a 500 mHz channel available, while in another you might have a 700 mHz one, and the RF system cannot adapt to accommodate both of them (antennas, front ends, etc). Weightless-W is ideal for use in the smart oil and gas sector, because there is likely TVWS available.

Weightless-N is ideal for sensor-based networks, temperature readings, tank level monitoring, metering, and more. **Weightless-N:** Weightless-N is an ultra-narrowband system that is very similar to SigFox. You can think of it as a LoRa Alliance-based version of SigFox. Instead of being a complete end-to-end enclosed system, it's made up of a network of partners. Weightless-N is ideal for sensor-based networks, temperature readings, tank level monitoring, metering, and more.

Weightless-P: Weightless-P is the latest Weightless technology. It offers two-way features and quality of service tiers. In terms of the underlying technology, it's 12.5 kHz channel is relatively narrowband. Is perfect for IoT solutions that require high density of end devices, long range, huge battery life, and true bi-directional communication.

Weightless N and P are the more popular options since Weightless-W has a shorter battery life and it has problems with the channel.

Chapter 3

Solution

3.1 Requirements

The widespread availability of computing technology is creating opportunities for interventions to promote environmental sustainability and ecological consciousness. Taking into account the main goal of our project we define the following set of requirements that our solution must fill:

Affordable

We as humans don't leave on the ocean, so most of the things that we develop are build to be used on land. Since this project aims for ocean monitoring we need devices that can withstand the adverse conditions of the oceans. The current solutions focus on building very expensive devices such as those shown on section 2.2.1, which have a certain number of redundancies, that allows them to be used with a certain confidence of success. The drawback of this solution is that by being expensive, its difficult to achieve a global coverage of the oceans. Our approach should focus then on minimizing the cost of the system to allow its massification.

Close to Real Time Data

We live in a world where everyone is connected, where technology is easily available to almost anyone. With this, people today expect that the information presented to them is in real time. That's why even with the limitations of the ocean environment, we should try to provide information as close to real time as possible.

Modular System

Nowadays with the huge increase in popularity and availability of IoT devices, multiple companies are developing numerous types of sensors. There are sensors to measure pretty much any type of thing and the prices are traditionally lower than the solutions that existed previously in the market. In order to allow this system to be implemented in a massive way, it should be able to accommodate a wide range

of sensors, so it needs to be implemented in a modular way so that the users can quickly change what type of sensors and data they will be recording.

Public Access API

Our solution has the main goal to gather information on whales and present it to both biologists and users. In order to achieve this goal we need to set a public access Application Programming Interface (API) that allows external applications to interact with both the devices and the server seamlessly. This API needs to be designed for: easy to use, easy to maintain, easy to extend and powerful enough to fulfill our requirements.

Battery Life

Since this system is to be implemented in the ocean where there are no sources of electric energy we should try to maximize the duration of the battery, to maximize the amount of results obtained.

3.2 Architecture

The Architecture of this project is the combination of three different components. The device, the server and the mobile application.

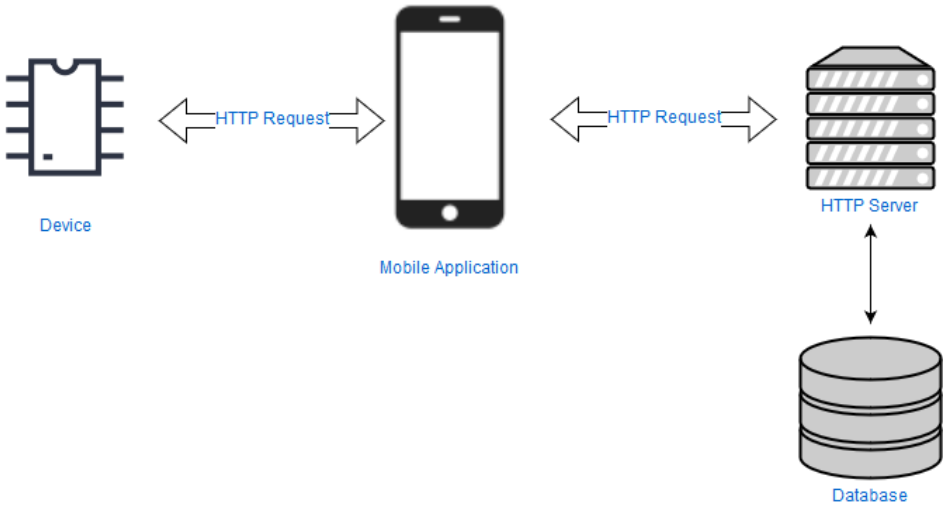

Figure 3.1: Project Architecture.

For the device we opt out for the Particle Photon, since we based some of the sound recording code from vasconcelos thesis. Another reason that made us choose this option was the fact that everyone in our department was using the same device, so it also made sense for us to use it.

The mobile application is implemented in iOS since is the platform that i'm most used to develop in. This application is the link between the device and the server since we don't have internet connection at the sea to allow direct communication between them. Since nowadays most of the communications between applications and servers are using Hypertext Transfer Protocol (HTTP) requests, it made sense

for us to also use them. Another advantage of this choice is the fact that they allow for the creation of a public API that makes the development of client and server independent.

Lastly, for the server we choose to use Node.js [32] for two main reasons. The first one is the fact that our requests are Javascript Object Notation (JSON) objects, which in reality are javascript objects, the language that Node.js uses, so choosing node for the server makes the server really simple. The last reason was the fact that at the proposal of this thesis professor Nuno Nunes already told that we were going to use Node.js for the server. To save the data that arrives on the server we choose to use a mongoDB[33] since is highly scalable and a non relational database that fits well with the type of data we are gathering.

3.2.1 Device

The device is a Particle Photon with some sensors that enables it to record environmental information. In terms of communication the device itself already has Wi-Fi capabilities that allows it to communicate with the user application. The device is placed inside a water resistant case along side the sensors. This case is essential because it allows the recording of data under water. Inside the case there is also a power bank that provides power to both the photon and the sensors.

Sensors

One of the objectives of this work was to build a system that was as modular as possible. In order to simulate a real world system this project includes some environmental sensors Table: 3.1.

Table 3.1: - List of sensors included in the device.

Sensors	Output	Ports
GPS	GPS coordinates	Rx-Tx
Bar100	Temperature, Pressure, Depth, Altitude	I2C
Turbidity Sensor	Turbidity	D6
pH Sensor	pH	A2
Dissolved Oxygen	Oxygen	A5
Microphone	Audio Recording	A1

The device only talks with the user application where it sends the information that has stored in the SD card, when requested. In order to facilitate the communication an API was created. This API needs to take into account some limitations that exists in the Particle Photon. In order to create a Wi-Fi network we need to use the device in listening mode by using the library SoftAP HTTP from particle [34]. This library creates a temporary access point (AP) and an HTTP server on port 80. The main objective for

this library is to allow the users to connect to the device, and give them the ability to configure the Wi-Fi access points that they want the device to attempt to connect. This library also allows the system to provide HTTP Uniform Resource Locator (URL) so that applications can add their own pages to the SoftAP HTTP server. Nevertheless this mode creates some limitations in terms of the size that the requests can have. After some tests conducted the sweet spot where we don't lose any information when answering a request is around 5KB. This raises a problem, because if the information in the file to send is larger than these 5KB, we need more than one request to obtain the information. At first we wanted to send everything in the JSON format to create the same API that we have on the server, but due to this limitation the alternative created was to save the data in Comma separated values (CSV) format, which is a way to condensate the information in comparison with the JSON format. Now, since we need a way to tell the client when all the information was received and what was the latest information that they received, we need to include two additional fields. the "dataPart" field, that tells the user what was the latest byte of information that they received from the file saved in the SD card, and the "isFinalData" field, that as the name suggests, tells the user if there is more information to send. In order to avoid a repetition in the information sent to the user we also added a field called "deviceId" that identifies from which device the user is getting the data.

In the figure 3.2 we can see an example of a response to a request done to the device. The URL of the device is always the same because it is acting as an AP. The URL is `http://192.168.0.1/data?0` where the 0 in the end is the latest "dataPart" that the client received. In this example since is the first request it starts at zero.

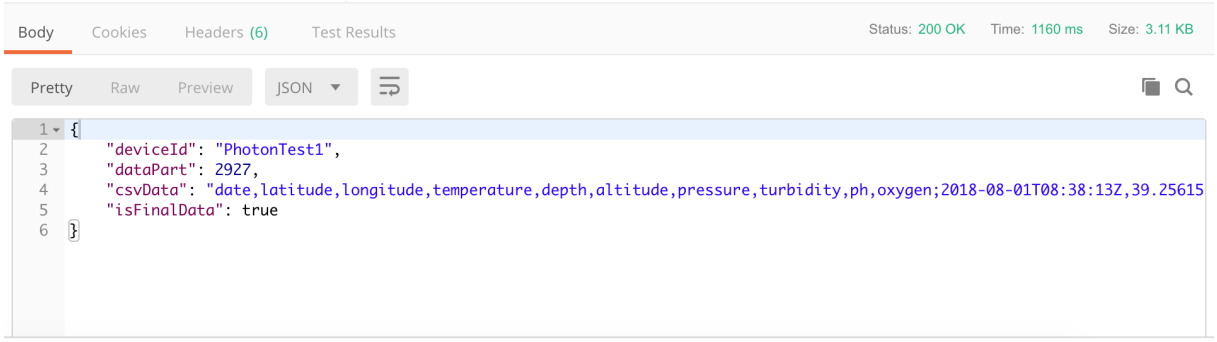

Figure 3.2: Example of a response to a request done to the device.

Due to the limitation in the size of the responses we decided that the audio recordings can't be send to clients because it will take to much time to send them all, and so we decided to just record and save them in the SD card. Later this information can be processed offline for any purpose.

3.2.2 Server

The server is built using Node.js [32] and provides an open API where everyone can retrieve the data recorded in the database.

The API that exposes the service is a REST API that uses HTTP requests to manage the data. This type of API has numerous advantages for the client applications and also the server because it's a

scalable, flexible and a portable solution. It's also a stateless solution meaning that each HTTP request has enough information, so that both the client and the server doesn't need to save any type of state. The main advantage of this solution is it's independence. Due to the separation between the server and the client it's possible to develop the various areas of the project independently. It's also possible to develop multiple types of clients that use the same API to retrieve the information.

The server API has two types of requests. A GET request to enable the retreat of information and a PUT request so that it is possible to put information to be kept persistently.

In order to save the data in a persistent way, we choose a NoSQL Database mongoDB [33]. Since in this project the data is actually recordings of sensors at a given time there is no need for a relational database. By choosing a NoSQL database and in particularly mongoDB [33] we gain numerous advantages such as:

1. Non-relational and schema-less data model
2. Low latency and high performance
3. Highly scalable
4. Object-oriented programming that is easy to use and flexible

Since we are using Node.js, we choose to use the Express framework [35]. Express is a web application framework that provides a robust set of features for web and mobile applications. The main advantage of this framework is that it offers all sorts of routing features including routing, separate handlers for *GET*, *PUT*, *POST*, and others, variables pulled automatically from URL and many more advantages that makes developing a web server an easy task. The URL (`http://localhost:8080/data`) for the request is the ip address of the machine where the service is executing, in this case is running on localhost, followed by the route path which in the case of our application is */data*.

3.2.3 User Application

In order to simulate a fully functional system, we developed a mobile application, for the iOS platform, which acts not only as an interface for the user to see the recorded data but also as a "mule" to help spread the information to everyone. The user application is the one that retrieves the information from the devices and sends it to the server, so that everyone can access it. The application was developed for the iOS 11.3 version and uses CocoaPods [36] in order to load all the external libraries needed. For a more friendly user experience we blocked the application from being used in horizontal mode, since the information that we want to show doesn't fit well in an horizontal screen.

To retrieve the data from the server and the device, the application uses the REST API provided by them. Since the information from the device is a JSON with a field named csvData with the format in CSV, the application needs to convert this information into an object model, so that it can be saved in the mobile device and sent to the server.

The user application mainly consists in 4 different screens and uses a UIPageViewController [37] to swipe between full-screen content pages. In fig. 3.3 we can see the flow of the application. Basically

to move to the next screen the user can press the button next or swipe to the left. To go back press the prev button or swipe to the right. At any moment the user can see in which screen he is by checking the dots in the bottom of the page.

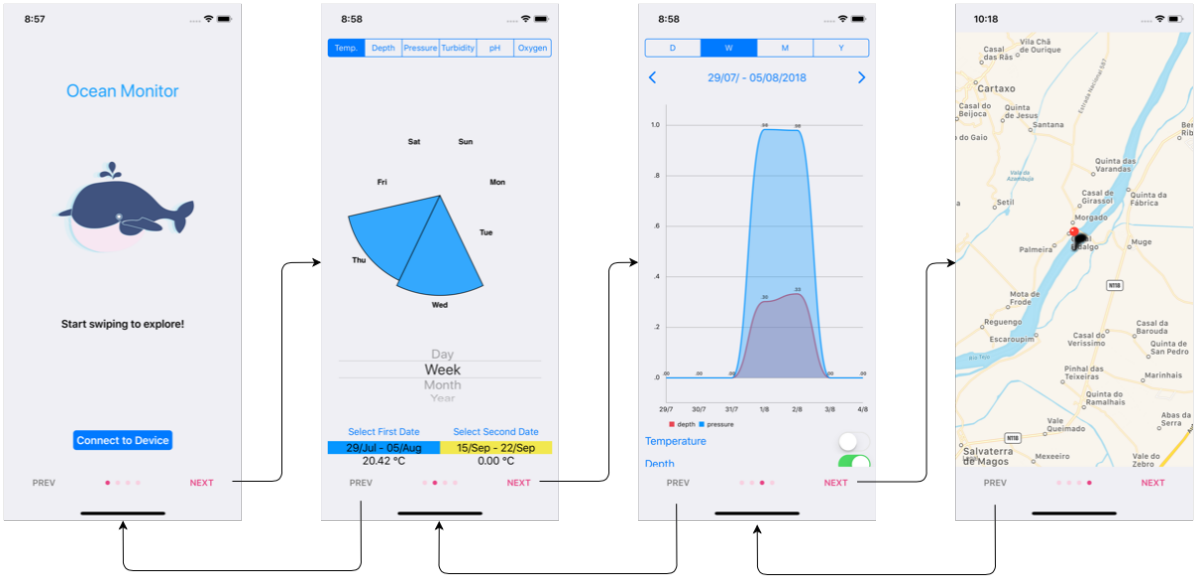

Figure 3.3: Mobile application flow diagram.

First Screen This screen is the Welcome page to the application. It allows the user to get the data from a nearby device if they are in the same local network. It also allows the user to refresh the application and get all the data from the server if there is internet connection.

Second Screen In the second screen the user can compare two different dates. There is a segmented control in the top of the screen so that the user can select what type of data they want to see. In the example of the available data the user can select between temperature, depth, pressure etc. There is a picker bellow the middle of the screen that allows the user to select if they want to compare by day, week, month or year. In the bottom of the screen and right on top of the page controller, the users can select what dates they want to compare and see the dates that they selected.

Third Screen In the third screen the users can see the evolution of the data at a given time. They can select up to three measures at one time to display in the graph. At the top of the screen they can select what type of view they want to see, if is an evolution of a specific day, week, month or year. The users can change the date by clicking on the arrows to increase or decrease the day. In the bottom of the screen the user can select what measures they want to see by enabling or disabling them in the correspondent switch.

Fourth Screen In the last screen the user can see the location where the recordings took place. It's also possible for the user to click on a given location and see all the recordings values as well as when

they took place.

The mobile application also stores the data locally so that it can work offline and send it to the server. In order to achieve this we implemented a database using Realm [38]. In iOS there are three alternatives to save information SQLite[39], Core Data[40] and Realm[38]. SQLite is an in-process library that implements a self-contained, serverless, zero-configuration, transactional SQL database engine[39]. Can be used in multiple platforms such as Mac OS, iOS, Android, Linux and Windows platforms. The main advantages of this solution are the no need for hassle of configuration, no server restrictions and easy and secure access to data from multiple threads. Core Data is a framework that you use to manage the model layer objects in your application[40]. It provides generalized and automated solutions to common tasks associated with object life cycle and object graph management, including persistence. Core Data actually stores the contents of an object which is represented by a class in Objective-C, that's why it uses more memory than SQLite and more storage space. On the other hand the biggest advantage of Core Data is it's speed where is faster in fetching than SQLite. Lastly we have Realm. Realm was designed to be faster and also more efficient than the previous solutions. It was designed to be multiple platform that's why database files can be shared between iOS and Android for example. By forcing the database files to work in the same way, and since our mobile app is the central point in getting data from the devices and uploading it to the server, we can make sure that in the future if we design an Android version of this application the data will be in the same format. Another big advantage of Realm, and probably the biggest one, is its simplicity. It takes very few lines of code to setup everything, compared to the other solutions.

3.3 Implementation

In this section we will describe in more detail the implementation of the project. In order to make it easier to understand the implementation description is divided in the same components as the architecture.

3.3.1 Device

The device while turned on can be at three different modes. The first one and the most basic is the idle mode. Due to the fact that the device needs at the start an internet connection to set the current time on the RTC, this mode is the initial one. To set up the time we use the SparkTime library[41]. This library creates a simple Network Time Protocol (NTP) client that allows the RTC to synchronize it's clock. The timezone that we selected to the current project is the Universal Time Coordinated (UTC) since it's a uniform standard time for legal commercial and social purposes.

The second mode is the SoftAP mode. While in this mode the device creates an Wi-Fi hotspot where everyone can connect and get the data using HTTP requests.

Last but not least, we have the recording mode. In this mode the device records audio and the values from the sensors. The audio recordings are constant, which means that is always recording and saving

the files in chunks of 4 seconds per file. The values from the sensors only get recorded if the timer defined by the user is surpassed.

In the implementation of the device we have one main file that controls everything *ParticleWhales-Thesis.ino*. In this file we can see two essential functions. The *setup()* and the *loop()*. When the photon starts running it always enters the *setup()* function.

```
// setup() runs once, when the device is first turned on.
void setup() {
  setupSerial(); //allows output in serial Monitor
  waitUntil(Particle.connected); // waitUntil network is available
  Serial.println("Connected to Particle ");

  // register the cloud function
  Particle.function("enableap", enableSoftAp); // function to start AP Mode
  Particle.function("enableread", enableReading); // function to start recording Mode

  // Needed Objects
  gpsSensor = new GPSSensor();
  timeManager = new TimeManager();

  //Initialize your Objects
  bar100Sensor = new Bar100Sensor(997); // kg/m^3 (freshwater, 1029 for seawater)
  turbiditySensor = new TurbiditySensor();
  phMeterSensor = new PHMeterSensor();
  dissolvedOxygenSensor = new DissolvedOxygenSensor();
}
```

Listing 3.1: Function setup() executed when the photon starts.

In the Listing 3.1 we can see that once the *setup()* function is called it will *waitUntil(Particle.connected)*. This method makes sure that the photon only continues its normal execution once it connects to the particle cloud. As already told we need this in order to make sure that we have internet connection to setup the RTC. We also register two functions in the particle cloud that allows us to switch the modes of the application. Since the SoftAp mode requires the usage of the Wi-Fi antenna we lose the connection to any network so we can't call in the particle cloud to disable this mode, only to enable it. In the end of this method we can see the initialization of the sensors. Note that in order for everything to work smoothly the GPS sensor and the RTC sensor (timeManager) must always exist.

```
// loop() runs over and over again, as quickly as it can execute.
void loop() {
  if(canRecord){
    recordingSoundLoop();
  }
  // checks if the current depth is less than 1 meter, if it is
  // disables the reading mode and turns the softAP mode on
  if(bar100Sensor->getDepth() > -1  && softAPMode == false){
    canRecord = false;
    state = STATE_STARTRECORDING;
```

```
    softAPMode = true;

    WiFi.listen(); // Enables Listening Mode

    softap_set_application_page_handler(myPage, nullptr);
    Serial.println("Now Listening: ");
  }

  // checks if current depth is bellow 1 meter and if softAP is on, if it is
  // disables the softAP mode and turns the reading mode on
  if(bar100Sensor->getDepth() < -1 && softAPMode == true){
    canRecord = true;

    softAPMode = false;

    WiFi.listen(false); // Disables Listening Mode
  }
}
```

Listing 3.2: Function loop() executed over and over by the photon.

In the Listing 3.2 we can see the implementation of the method *loop()*. This method checks three conditions. The first one is to check if the *Boolean* variable *canRecord* is true or false and consequently call the *recordingSoundLoop()* function. The two other conditions have the opposite purposes. While the second one checks if it can enable the softAP mode, the third one checks if it can disable it. They both rely on the *Boolean* variable *softAPMode* to check if it's enable or disable.

The reason why we cannot be in the record and softAP mode at the same time, is because in recording mode we are constantly accessing the SD Card to save the audio recordings, so to avoid conflicts while accessing it, the two modes can't work together. In terms of development it also helped us because it separates two different concepts that can be developed independently. If there is no need to save the audio information, the device can always be in softAP mode and before writing new values to the SD Card we only need to disable that mode to avoid connections and consequently conflicts. We will now talk more in detail about these two modes since they are a key feature of the project.

SoftAP Mode

The softAP mode when activated runs on a different thread than the main program execution, that's why we have to take into account the concurrency problems that might occur. Another big problem is that always enabling the Wi-Fi also increases the battery consumption, which is one thing that we want to avoid.

```
void myPage(const char* url, ResponseCallback* cb, void* cbArg, Reader* body, Writer*
    result, void* reserved);
```

Listing 3.3: softAP handler declaration.

The Listing 3.3 shows us a declaration of an handler *myPage*. This handler will then be called every time a client makes an HTTP request on the URL *http://192.168.0.1/*. The *url* argument in the function is the path of the file requested by the client, and doesn't include the server name or port. In our application the typical *url* will be */data?0*, where the zero is the last *dataPart* received by the client. In this case since it's zero is the first time that the client is making a request. The *cb* argument is a response callback, and it's used by the application to indicate the type of HTTP response, 200(OK) or 404(not found). The *cbArg* is data that should be passed as the first parameter to the callback function. The body is an object that the page handler uses to retrieve the HTTP request body. The result is a writer object that the handler uses to write the HTTP response body. The last argument *reserved* is there for future expansion of the softAP library and for now always equals to *nullptr* [34]. Our handler implementation after verifying that the request is valid, calls another function that receives the *dataPart* value and starts reading the data file at that position. Once the limit of the 5KB is achieved it closes the file and generates the JSON String that is sent as a response to the client. As previously mentioned in the 3.2.1 section the JSON contains the *isFinalData* field to tell the client if this is the last part of the file.

Recording Mode

The recording mode can also be divided in two components. The recording of the audio and the recording of the sensors. In terms of the audio recording we implemented the version developed by Dinarte Vasconcelos [15] with a few changes in the code because we don't send the recording immediately to the server and instead we just saved them to the sdCard. In this mode we have three different states, the *STATE STARTRECORDING* where we initiate the recording of the audio file. The *STATE RUNNING* where we check if one of the two buffers is full and if it is, save it to the sdCard, and it also checks if the time of the recording already exceeded the defined time. Lastly we have the *STATE FINISH* where we finish the recording and also check if it's time to read the sensors.

Since nowadays there is a wide variety of sensors, and we want this device to support all of them, while keeping it simple to add and remove sensors, we decided to enforce that every sensor must implement an interface called *SensorsInterface*, Listing 3.4. This *SensorsInterface* is an interface with only two methods. A *record()* method that is used to record the value and keep it in a variable at the choice of the user and a *getRecordValue()* method that returns a String with the value that the user wants to save. With this implementation we force the user to create a *class* for every sensor which implies that the code for that sensor will only live inside that class, which makes the application independent from the sensors, and the sensors independent from themselves.

```
class SensorsInterface {
  public:
    // record the value from the sensor for later use
    virtual void record() = 0;

    // gets the value from the sensor as a string.
    // If you want to return more than one value for a specific sensor they should be
    // separate by a comma
```

```
    virtual String getRecordValue() = 0;
};
```

Listing 3.4: SensorsInterface.

Although this interface is very simple, it is actually very powerful if it's used correctly. However, in some cases, the code for the sensors might not be that easy to implement, and that's why we only force these two methods to be implemented. For instances, in our case the *DissolvedOxygenSensor* actually needs the current temperature in order to work, so we added one method to set the temperature value for the sensor. There is also another case where the *Bar100Sensor* wants to save multiple values, so it's possible to return a string with all those values separated by a comma, so that they can be saved in the CSV file. The main purpose for us to follow this approach, was to keep the code of the sensors as independent as possible in order to allow its easy addition and removal. In the *config.h* file there is a parameter called *csvInitialString* where the users should specify the beginning of the CSV file. The users must then respect the order of this string when getting the sensors values.

3.3.2 Server

The server is a HTTP server that uses the Express framework [35]. In the Listing 3.5 we can see the beginning of our server application where the constant *app* is the express object that is defined in the file *./app.js*. Using the HTTP built in Node.js we start the server on the port 8080, or in the port given by the cloud provider.

```
const http = require('http');
const app = require('./app');

const port = process.env.PORT || 8080;

const server = http.createServer(app);

server.listen(port);
```

Listing 3.5: HTTP Server.

In our application *api* folder we have the *models* and the *routes*. Since our application at the moment is a small one, we only have one of each other, but for future expansion it's possible to add more routes and models. Our *dataModel* is a *mongoose* schema. Mongoose is a mongoDB object modeling for Node.js and allows us to save our data to the database and retrieve it. In the Listing 3.6 we can see our dataSchema with all the fields for the sensors information and their respective types. The *id* field is a concatenation of the deviceId and the date that allow us to create a unique identifier for every device recording across the server, mobile application and devices. The date is saved in ISO8601 [42] format as a string. We use this format to standardize so that every application can convert to and from it, and once more allowing the server to be independent from the data.

```
const mongoose = require("mongoose");
```

```
const dataSchema = mongoose.Schema({
  _id: String,
  deviceId: String,
  date: String,

  latitude: Number,
  longitude: Number,

  temperature: Number,
  depth: Number,
  altitude: Number,
  pressure: Number,
  turbidity: Number,
  ph: Number,
  oxygen: Number
});

module.exports = mongoose.model("data", dataSchema);
```

Listing 3.6: Data model schema.

Right now we have a simple API allowing only a *GET* and *POST* request, and in the *GET* request we always send all the available data. If required by the client applications this API can be easily extended, and for example only send information about one specific device given the respective *deviceId*. Even though on this project we didn't implement any security measures, such as integrity of the data so that it couldn't be changed by unknown entities, the server always check if the data received supports the *dataSchema* presented above. If not it's immediately rejected by the server.

3.3.3 User Application

In iOS mobile development, the main pattern used is the Model View Controller (MVC). This pattern is widely used for developing user interfaces that divides an application into three interconnected parts. In iOS we have three types of objects with different roles: model, view, controller[43]. Each one of the three types of objects is separated from the others by abstract boundaries and communicates with objects of the other types across those boundaries. In figure 3.4 we can see an example of how the communication is handled using this separation of concepts.

Model Objects Model objects encapsulate the data specific to an application and define the logic and computation that manipulate and process that data. The model should be independent of everything and rule the application by directly managing the data and logic. The model only communicates with the controller and typically is an object that the controller manages.

View Objects A view object is an object in an application that users can see. Basically it can be any output representation of information and it's also responsible to respond to user actions, by sending

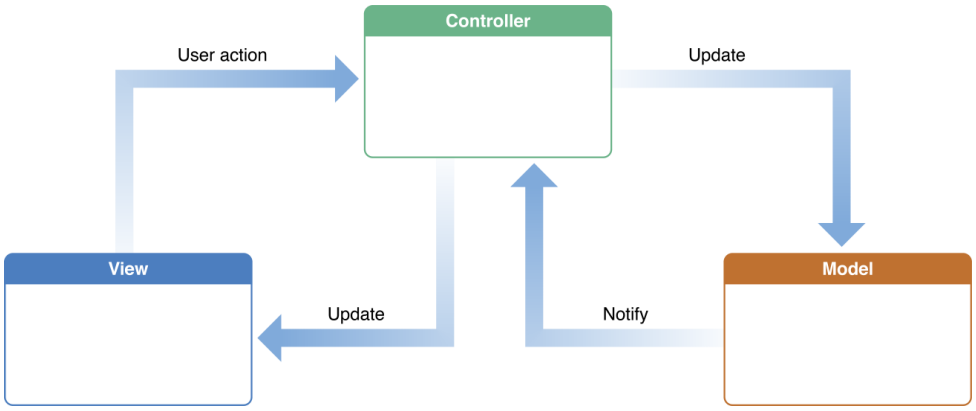

Figure 3.4: MVC example.

an information to the controller. You can have multiple views with the same information but different interfaces. Normally a view only interacts with the controller.

Controller Objects A controller object acts as an intermediary between one or more of an application's view objects and one or more of its model objects. Controller objects are thus a conduit through which view objects learn about changes in model objects and vice versa. Controller objects can also perform setup and coordinating tasks for an application and manage the life cycles of other objects.

In iOS the controller object is called as a *ViewController* and it has an important lifecycle. The iOS calls the lifecycle methods at different stages. The main important one for us in this thesis is the *viewDidLoad()*. This method is called when every component from the view is loaded from the storyboard. If we want to populate our view with data, this is the best timing for us to fetch all the data and give it to the components. Because of this, this is the most important and the mainly used lifecycle method. Other methods like the *viewDidAppear()* or *viewWillDisappear()* are used when the view appears on the screen and when the view is going to be removed from the screen respectively.

The user application is responsible to show the information obtain by the devices in an easy to understand way. In order to facilitate the usage of the data we created a model that takes care of all the conversions for us. Since we are using Realm [38] we can also use this model to directly fetch and save the information in the database. This model is implemented as a *class* in swift, in the Listing 3.7 we can see the optional initialization used when receiving data from both the server and the devices.

```
class DataModel: Object {

    /// init of the object used when receiving from server and device
    convenience init(isFromServer: Bool, deviceId: String, date: String, latitude: Double,
        longitude: Double, temperature: Double, depth: Double, altitude: Double, pressure:
        Double, turbidity: Double, ph: Double, oxygen: Double) {
        self.init()

        self.id = deviceId + date
```

```
        self.isFromServer = isFromServer

        self.date = DateFormatter().stringToDate(str: date)!
        self.deviceId = deviceId

        self.latitude = latitude
        self.longitude = longitude

        self.temperature = temperature
        self.depth = abs(depth)
        self.altitude = altitude
        self.pressure = pressure
        self.turbidity = turbidity
        self.ph = ph
        self.oxygen = oxygen

        isNull = false
    }
```

Listing 3.7: DataModel schema object.

If we go back to subsection 3.2.1, and remember that one of the problems that the devices had was the limit of size in the HTTP responses, we know that in order to get all the data we need to do multiple requests until we get the JSON field *isFinalData* with a *true* value, indicating that this was the last data. With this in mind and in order to facilitate the communication, we separated the connection between the server and the device, since the requests and treatment of the data is completely different, in two objects that can be called independently and handle all the work automatically. In order to facilitate the HTTP requests we used the Alamofire library [44] for iOS. The main objective of Alamofire is to simplify HTTP networking in iOS. It carries out its mission with incredible efficiency by harnessing NSURLSession and the Foundation URL Loading System to create a networking interface that feels native to swift.

Starting with the connection between the application and the device, we created a class called *ConnectionToDevice*. This class has one public method *getData(i: Int)* that receives an integer, indicating the last data received. This method works in a recursive way calling it self and adding the information to a dictionary until it gets the value of *isFinalData* true. When the field is true, the function *convertCSVDataAndSaveToDb(deviceId: String)* is called and starts converting all the information saved in the dictionary based on the device identifier. As already told the device sends the data in CSV format so the application transforms it to JSON format, and then, we initialize a *DataModel* object for each recording. After the initialization of the object we call our *DbConnection* object to save the recording into Realm.

In order for this to work the mobile device needs to be in the network created by the photon. Since we are using a free developing account we can't use the class *NEHotspotConfiguration* [45] that would allow us to automatically join any network created by our devices, so we added a button in the end of the main screen page that allows the user to retrieve the data if they are connected to a photon. Unfortunately due to the limitations of the free tier in the development accounts we can't automate this process, but if

this app is to be deployed on the appstore, this can easily be done.

Moving now into the connection between the application and the server, we will have two different types of requests. A *PUT* and a *GET* request. In the *PUT* request the mobile application sends all the information that it has received from a photon device and that has not yet been sent to the server, while on the *GET* request the application receives the information from the server and saves it to the database. Once again we created a *class ConnectionToServer* that has two public functions, *sendDataToServer* to send the information and *getDataFromServer* to get the data from the server. These two functions are always called when the main screen page loads, which means that they are called on the *ViewDidLoad()* method.

In the section 3.2.3 we briefly explained what the different screens were doing. We will now explain in greater detail how each one of them was developed.

We already know that usually for each view we will have a different view controller that will manage everything inside that view. So since we have four screens we expect to see four different view controllers in the Main Story Board (figure 3.5), but we actually have five. The first view controller loaded by the iOS is the one that has the arrow at the beginning pointing at him, which in our case is the *PageViewController*. This controller is the one responsible for the changes between screens and that's why we don't see any connection in the Story Board between the controllers. All this interactions were done directly using code instead of the interface builder since it's much easier to implement, and also less susceptible to errors. This *PageViewController* extends the *UIPageViewController* class and implements two protocols that enables it to respond to navigation gestures and to load the next view controller. The base implementation doesn't include the previous and next button so we added two functions that set view controllers when they are touched. The *PageViewController* view has a *bottomControlsStackView* that contains the two buttons and the page controller and because they belong to the main view they appear in every screen. The *PageViewController* calls an instance of the next view controller when a change occurs, by calling the method *viewControllerInstance* that instantiates a view of the corresponding view controller.

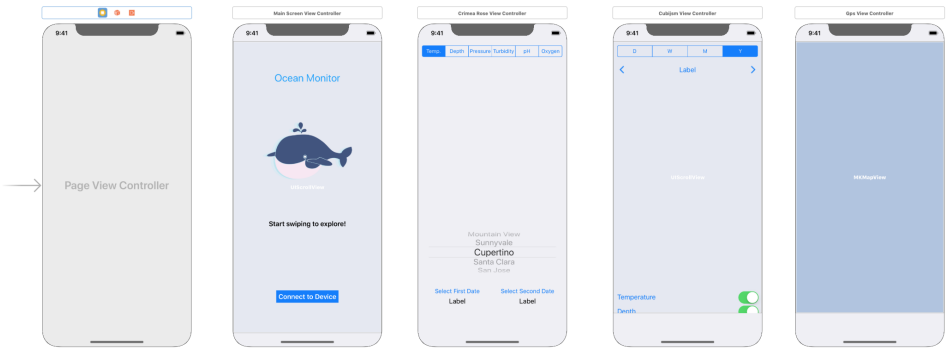

Figure 3.5: Main Story Board.

The first screen fig: 3.6 is immediately called by the *PageViewController* and is the first one to be loaded. Because of the page controller we added a constrain not allowing anything to be drawn in the last 50 points of the screen. In terms of components this screen only has a button to try to connect to a device and a label telling the user if they have information to view. There is also a refresh controller that allows the user to refresh the connection to the server in order to download information and update their database.

In the second screen we were inspired by the visualization Nightingale's Rose [46] and since there was no library available for iOS that could do that, we decided to implement it ourselves. In order to draw everything we used the UIKIT framework [47] from apple. To make it possible we started to plan how we would make it. Our design solution starts by dividing a circle between the number of items that we want to place. For example a year will have 12 semi circles, one for each month, so we need to divide the circle in 12 equal parts. The easiest way to achieve this is to divide the degrees of a circle, 360 degrees, by the number of parts we want. This division will give us an angle that we can use to draw. Thankfully the UIKIT framework has *UIBezierPath* class that allows us to draw lines on the screen. This class has a method that creates an arc (Listing: 3.8) and we then just need to add a line from the arc to the center of the circle and close the path, that it will automatically connect the other side of the arc to the last point of the line, which is the center of the circle. And by creating this simple object we have a semi circle. We then do exactly the same thing to every other semi circle that we want to draw.

```
let path = UIBezierPath()

path.addArc(withCenter: CGPoint(x: bounds.midX, y: bounds.midY), radius: radius,
    startAngle: startAngle, endAngle: endAngle, clockwise: true)
path.addLine(to: CGPoint(x: bounds.midX, y: bounds.midY))
path.close()
```

Listing 3.8: UIBezierPath method addArc.

The only think that varies between the circle is the radius and the start and end angles of the arcs. In order to achieve this we need to scale all the data with the information that we have. To scale, we need to receive the minimum and maximum values of the data that the user wants to see. With this information we make a scale and apply that to all the data. We choose this approach to allow the user to customize the visualization. Another thing that the users need to take care is the labels that they want to show. This labels must have the same number of items, otherwise it will have an error, since we can't have more labels than items and vise versa. To facilitate and standardize the way to send the data to the visualization we created a structure *CrimeaRoseData*, that can be seen in the Listing 3.9. This structure makes possible to the visualization to receive any number of different elements at the same time. In our case we will only use two dates, represented by the colors blue and yellow, but the visualization can handle any number.

```
struct CrimeaRoseData {
    var arrayOfData = [Double]()
```

```
    var color = UIColor()

    init(arrayOfData: [Double], color: UIColor) {
        self.arrayOfData = arrayOfData
        self.color = color
    }
}
```

Listing 3.9: CrimeaRoseData.

One key aspect of our view is that it is responsive to the user so it can handle a few movements. We added four gesture recognizer. The *UIPinchGestureRecognizer* lets the user zoom in and out in the view allowing for an easy understanding. The *UIPanGestureRecognizer* allows the user to move the image in the screen, which is useful when combined with the pinch to zoom gesture. Then we have the *UIRotationGestureRecognizer* that makes possible the rotation of the view, and last but not least we have the *UITapGestureRecognizer* that recognizes the semi circle that is touched and shows their values.

To complement the visualization we added a segmented control to allow the user to choose what type of data they want as well as a date type picker so that the user can select in what temporal order they want to compare the data. In the bottom of the screen we added two buttons that allow the user to select the dates that they want to compare. When they are touched it appears a pop up with a date time picker that allows the users to select the date. There is also two labels that tell the user what date they choose and the correspondent color. Bellow this there are two other labels that show to the user the values of the semi circles when they touch them.

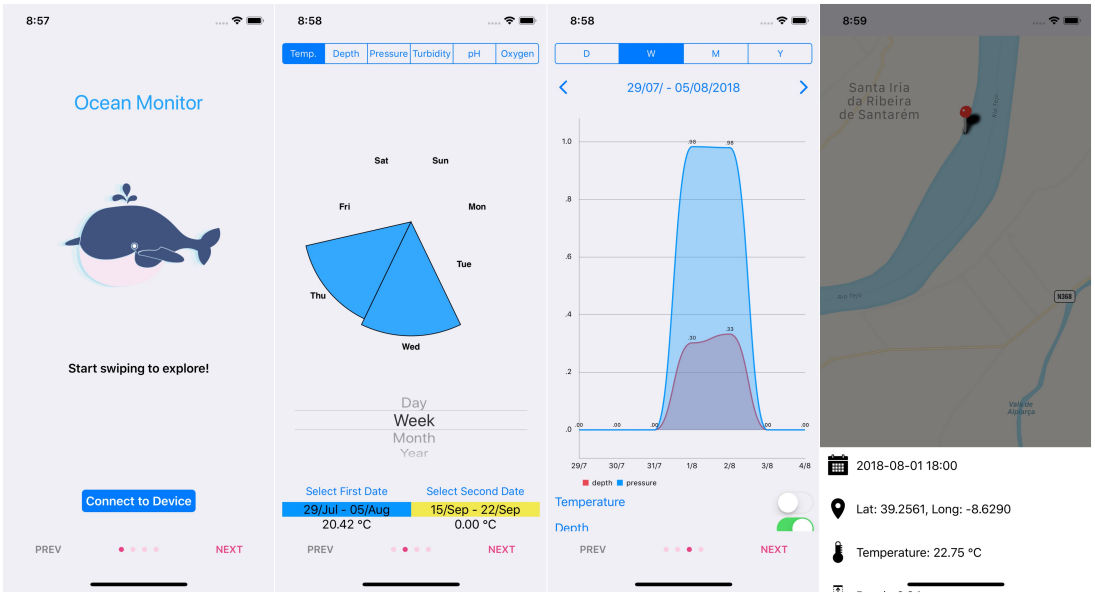

Figure 3.6: Screens images from left to right: first, second, third and fourth screen

The third screen is implemented inside a *UIScrollView* [48]. The central notion of a UIScrollView object (or, simply, a scroll view) is that it is a view whose origin is adjustable over the content view. This

means that we can place more content inside the view that it will automatically allow the user to scroll to see all the information. In our case our view allows the user to scroll vertically and see all the content. In this view the main objective is to allow the user to check the evolution of the data over time. With that in mind we decided to implement an Area chart. An area chart is actually a line chart but that is filled. For iOS there are already some libraries that allow us to create some charts without having to draw everything by ourselves. For this project we choose the library *charts* [49] created by Gindi. The main advantage of this library is that it allow us to freely customize the chart we want to draw. Every chart drawn with this library has right away animations, dragging, panning and scaling. In order to implement this graphic in a view we instantiated an object of the type *LineChartView* and added it as a subview of our main view. The *LineChartView* as an attribute data of the type *ChartData* that needs to be set in order to display the information. In our case since we are drawing in a line chart we used the object *LineChartData* which extends the *ChartData*. The user has the possibility to choose if they want to see the data by day, week, month or year by selecting the option in the segmented control at the top of the screen. If the user scroll down they can see numerous switches. Each switcher makes visible in the graphic the data to the correspondent attribute. Because of restrictions with the size of the screen we only feel comfortable in showing at maximum three attributes at the same time, so if the user selects another one it automatically deselects one of which was being shown. To select the date the user has to click on the arrows bellow the segmented control and the view will automatically refresh and display the new data.

In the forth screen we used the apple MapKit framework [50] to visualize the GPS coordinates recorded by the device. For each location recorded we added a *MKAnnotation* which creates a pin in the map. The map automatically aggregates the pins if they are to close to each other, and shows them when the user zooms in. We created a pop up that appears when the user touches a pin, and gives all the information about that recording like the time, location and sensors data. In order to create this pop up we create a *PopupLauncher* object that has a *collectionView* which in turn has any given number of cells. This cells are just a model object that contains a name of an icon and the value to show.

Chapter 4

Results

Our work comprises the development of a system that could sustain the adverse conditions of the oceans, while maintaining the requirements described on the section 3.1. Due to the difficulty in deploying the device in the ocean we opt to put it on the Tejo river near Santarém. For safety reasons we didn't submerge the case, we left it floating on the river, and because of this we didn't test the mechanism of switching between the recording mode and the SoftAP mode since the water level was always above one meter.

While we didn't have the sensors to put in the device we manage to develop all the connection infrastructure that connects the device, the mobile app and the server. With this infrastructure we forced the API to be respected or otherwise the applications will return errors. This errors saved us a lot of time because when we switch the API to accommodate all the sensors, if we make any mistake the applications will immediately send errors that allows us to easily detect and fix them.

Figure 4.1: Exterior of the prototype case.

4.1 Hardware Prototype

As already said, in order to simulate a real environment we installed some sensors in an particle photon device. The list of the sensors can bee seen in the table 3.1. We used a peli 1150 case [51] fig.4.1 as a box containing all the sensors, power bank and the device. The case by itself is already waterproof but since there are some sensors that need to be outside the case we had to drill some holes. The turbidity sensor needs to be in direct contact with the water and doesn't have a large cable so we did a hole in the bottom of the case just for this sensor. This sensor can be seen in the fig. 4.1 as the transparent sensor in the bottom of the case. The Dissolved Oxygen and the pH sensors can be connected with a cable so we made two holes on the top of the case that allow us to connect the cables, and if we are not using them we can simply detach them. The Bar100, the red sensor in fig. 4.1, has a long enough cable that allows the hole to be on the top of the case while the sensor still reaches the water. In order to power the device we placed a 10000 mhA powerbank inside the case. All the other sensors are inside the box and don't have any contact with the exterior. In fig4.2 we can see the interior of the case. The powerbank can be placed in the middle of the case, in the unused space that can be seen on fig. 4.2. On the left side of the case in fig. 4.2 we have the microphone, a litle bit to the left we have a yellow square that is the GPSsensor. On the top of the case we have the connectors for the Dissolved Oxygen and the pH sensors. The last equipment that can be seen on the fig 4.2 is the particle photon were every sensor connects.

4.2 Data Gathering Simulation

As told in the beginning of this chapter we placed the device in the river in order to simulate the conditions of the ocean. In the first test we left it there for more than 24 hours until we retrieve it. On the first test everything went as expected since we had already test it. In terms of the sensors data we had problems with the pH sensor and it's calibration. The battery lasted for about 23 hours. In the photon we can't turn off the power to some ports, and the device even if in sleep mode, keeps providing energy to all the ports. With so many sensors connected the battery can't last longer. Unfortunately we can't solve this issue with the photon, because one possible solution is to have a switch managing the power given to the photon, and that would wake up the photon, let it record and save the data and then power it off. The problem with this solution is that we need to have internet every time the photon wakes up to set the rtc. With this solution we also couldn't be always listening with the microphone.

After this try we made changes to the pH sensor to improve the results. We made four tests were every test went as expected. All the other tests duration were shorter because of the restrictions with the place.

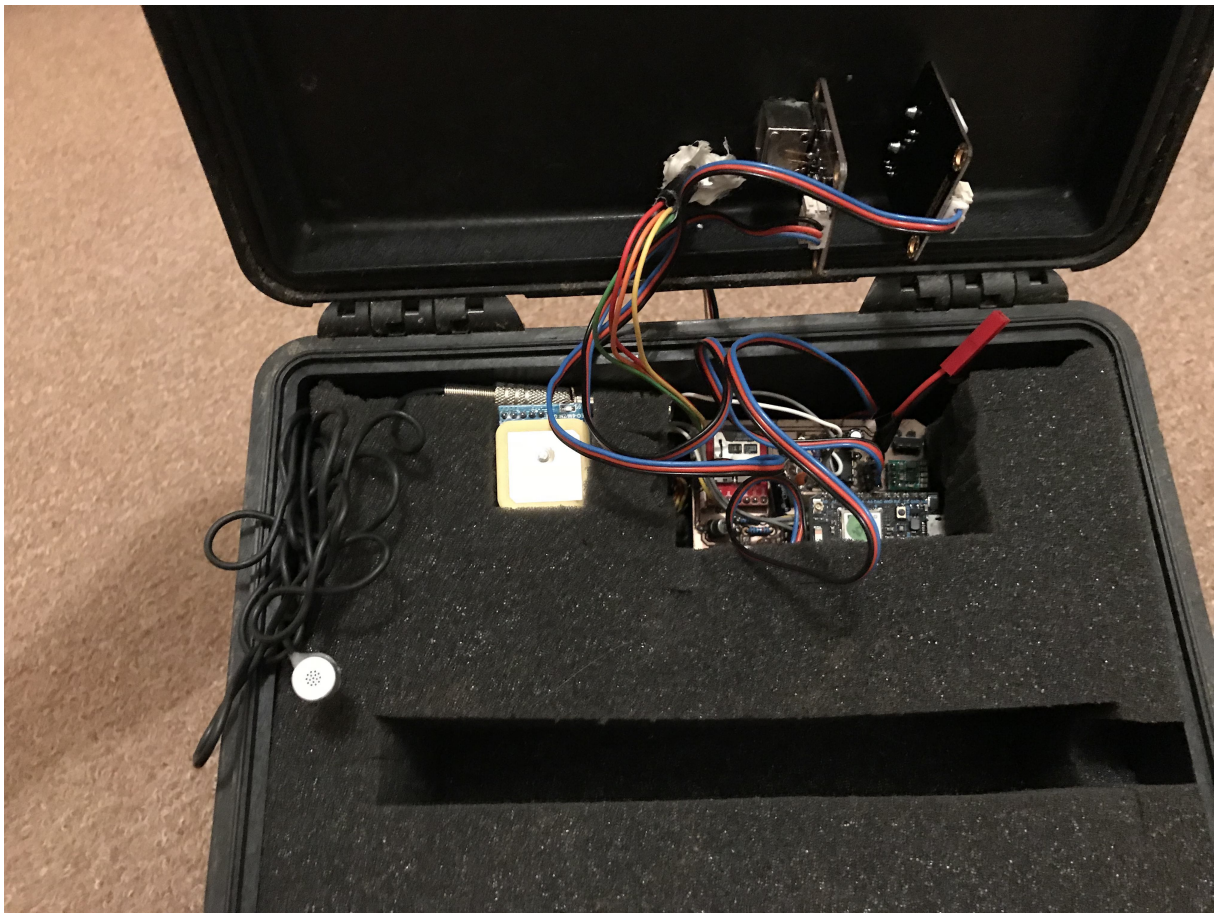

Figure 4.2: Interior of the prototype case.

4.3 Device Requests

In order to test how would the device behave in a real situation, were we can have multiple people getting the same information at the same time, we conducted some tests. We choose the software jmeter [52] from Apache to conduct those tests. Jmeter gives us the ability to do load and performance tests on many different applications. In our case we are interested in the ability to test the requests from the device. In the first test we choose to simulate a single user doing 10 requests to simulate a user getting a large file of data. In the fig. 4.3 we can see that in average a request takes about 600 milliseconds and for a total of 10 requests it took 6 seconds to get all the answers.

For the next test we tried to simulate two concurrent users each making 10 requests each. In the figure 4.4 we can clearly see that the average time for each request to be answer doubled to almost 1 second.

In spite of the fact that the time for each request doubled, we can see other problems if we look closer to the figure 4.5. In order to better simulate a real world application we made some restrictions to the requests. The biggest one is the timeout of 2 seconds for each request. With this restriction we can see that in the column status two of the requests didn't arrive since the status isn't ok. This results look good if we don't look to the bytes column where it shows us the actual number of bytes that we received. Since we know that the length of the answer is about 4 kB we can assume that a correct answer should

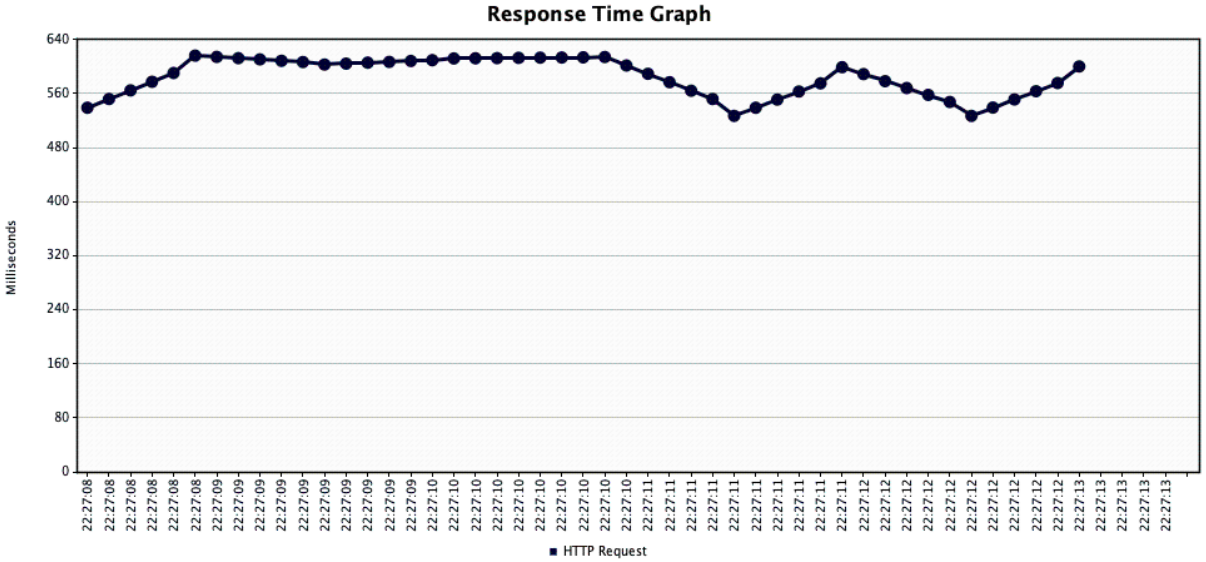

Figure 4.3: Response time graph for 10 requests made by a single user.

have the number of bytes equal to 4133. With this information we can see that half of the requests made aren't correct so we can conclude that the device can't handle two users at the same time.

There are two big reasons for the problems with multiple concurrent users. The first one is the limitations on the photon SoftAP mode that wasn't made for this kind of application. The second one and probably the one that limit's the most is the fact that every request is trying to access the SD card, and every request is on a different thread that we can't control. With this we will have concurrency problems where two users are trying to get information from different parts of the SD card at the same time.

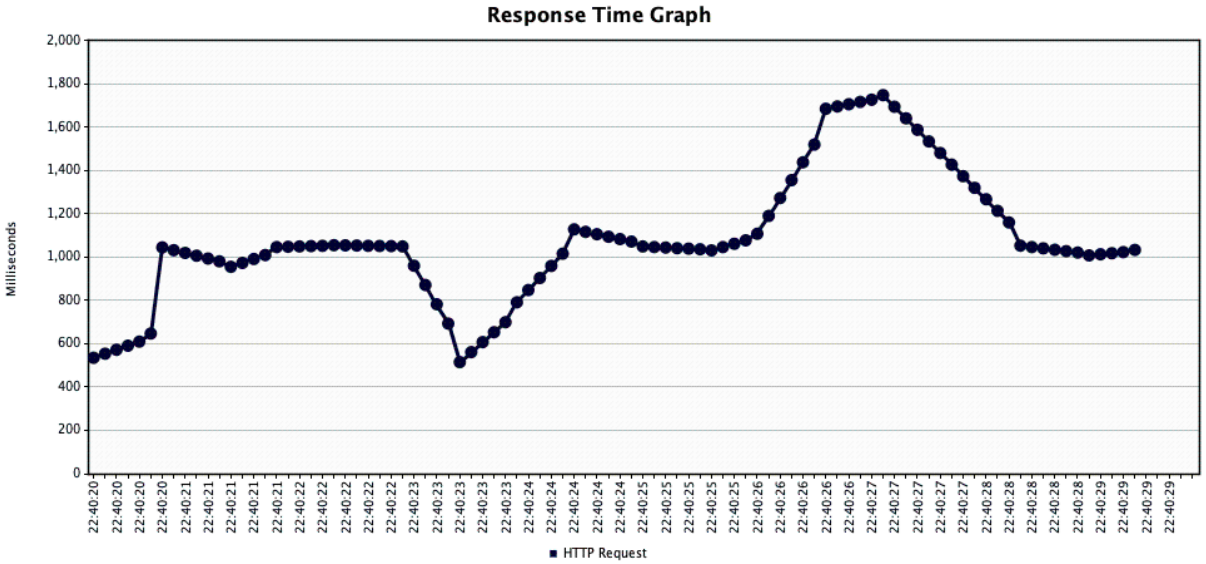

Figure 4.4: Response time graph for 10 requests made by two concurrent users.

Sample #	Start Time	Thread Name	Label	Sample Time(ms)	Status	Bytes	Sent Bytes	Latency	Connect Time(...
16	22:40:27.264	Group 1 1-1	HTTP Request	1748	✓	224	119	1748	1171
15	22:40:26.753	Group 1 1-2	HTTP Request	1685	✓	4133	119	1681	1008
20	22:40:30.020	Group 1 1-1	HTTP Request	505	✗	2172	0	0	135
1	22:40:20.381	Group 1 1-1	HTTP Request	535	✓	4133	119	533	7
12	22:40:25.105	Group 1 1-1	HTTP Request	1049	✓	4133	119	1046	7
2	22:40:20.882	Group 1 1-2	HTTP Request	647	✓	4133	119	554	3
9	22:40:24.056	Group 1 1-2	HTTP Request	537	✓	224	119	537	3
7	22:40:23.540	Group 1 1-2	HTTP Request	515	✗	2172	0	0	2
8	22:40:23.010	Group 1 1-1	HTTP Request	1049	✓	4133	119	1045	2
19	22:40:29.492	Group 1 1-2	HTTP Request	1033	✓	224	119	1033	1
3	22:40:20.917	Group 1 1-1	HTTP Request	1045	✓	4133	119	1044	0
4	22:40:21.530	Group 1 1-2	HTTP Request	955	✓	4133	119	953	0
5	22:40:21.963	Group 1 1-1	HTTP Request	1046	✓	4133	119	1044	0
6	22:40:22.485	Group 1 1-2	HTTP Request	1055	✓	4133	119	1045	0
10	22:40:24.059	Group 1 1-1	HTTP Request	1046	✓	224	119	1046	0
11	22:40:24.593	Group 1 1-2	HTTP Request	1128	✓	224	119	1128	0
13	22:40:25.721	Group 1 1-2	HTTP Request	1031	✓	4133	119	1024	0
14	22:40:26.155	Group 1 1-1	HTTP Request	1108	✓	4133	119	1102	0
17	22:40:28.438	Group 1 1-2	HTTP Request	1053	✓	224	119	1053	0
18	22:40:29.012	Group 1 1-1	HTTP Request	1008	✓	224	119	1008	0

Figure 4.5: Table showing results 10 requests made by two concurrent users.

4.4 Server Requests

In order to evaluate the server and database performance we conducted a load test for them. In the table 4.1 we can see the specifications of the machine were the tests were conducted.

CPU Cores	2
Clock Frequency	2,7 GHz
RAM	8 GB
Storage	120 GB

Table 4.1: - Test machine specifications.

The tests were conducted on the same machine of the server, so the address used was `http://localhost:8080/data`. To test we used the jmeter [52] software. We stress test the server with 100 concurrent users each one doing 100 requests. All the requests were answered correctly by the server and the test duration was 1 minute and 14 seconds. To compare we did the same tests but changing the number of concurrent users. The results can be seen in the table 4.2

	1 User	50 Users	100 Users
Samples	100	5000	10000
Average Response Time	10ms	380 ms	730 ms
Error %	0 %	0 %	0 %
Throughput	95.3/sec	127.5/sec	134.4/sec
Received	1121.8 KB/sec	1500.44 KB/sec	1581.41 KB/sec
Sent	11.36 KB/sec	15.19 KB/sec	16.01 KB/sec

Table 4.2: - Server stress test.

After viewing the table 4.2 we can see that the server can answer all the requests since there are no errors. In terms of the average response time, as expected, the time increases with the number of concurrency users since there are more requests and the server can't answer them all at the same

time. This average response time is influenced by the fact that we are conducting the tests in the same machine of the server, so if we deploy the server to a cloud provider, for example, we will have significantly higher times. The throughput of the server slowly increases since, with a higher number of concurrent requests, we are feeding the server with more requests per second so we will maximize the number of requests that are processed in any given instant.

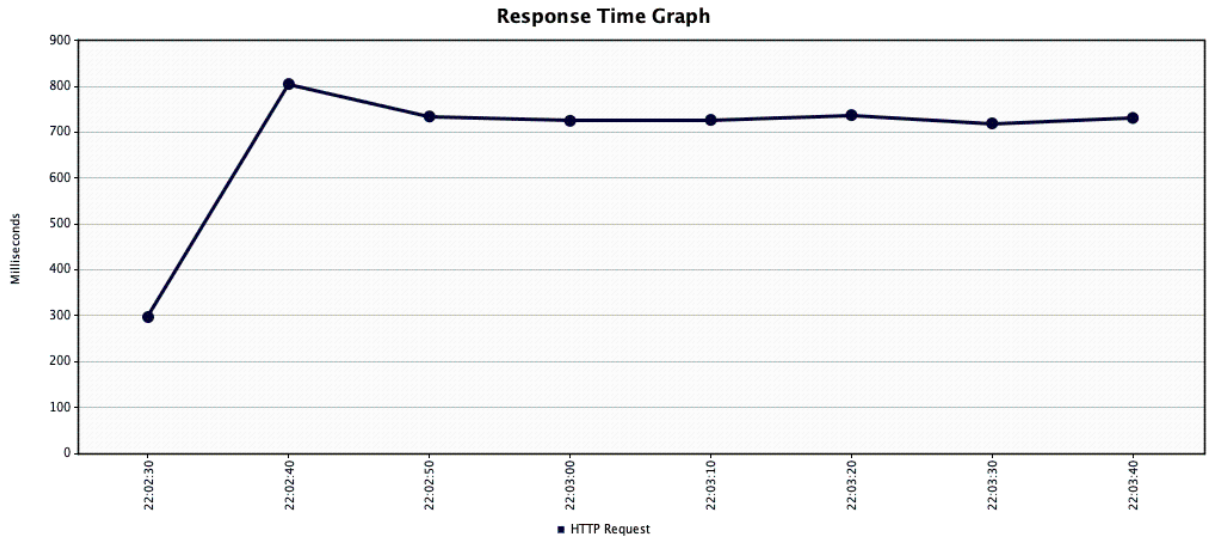

Figure 4.6: Graph showing the average response time for a request when we have 100 concurrent users.

In the figure 4.6 we can see that the response time increases rapidly until it stars to stabilize around the 730 ms for each request. This increase occurs because when the jmeter simulates 100 users it takes some time to launch all the 100 correspondent threads which is why we see an increase in the response time, as time goes by.

4.5 Real World Test Scenario

In order to simulate the real world system we conducted a test where every component of our project works together. The objective of the test was to get all the information into a user that was never close enough to the device to collect the data. In order for this to be possible the information in the device needs to be collected by another user with the use of the mobile application. The information needs then to be sent to the server by this application. To conclude the test the initial user needs to open his mobile application with internet access and get the data from the server. The test succeeds if the information in both users application is the same.

To conduct this test, and since we can't use more than one device to install the application we used a real iPhone 7 as the user that gets the data from the device, and the iPhone X emulator available in macOS as the user that can't get close enough to the device and can only get the information from the server.

In figure 4.8 we can see the result obtained after performing the simulation test. On the left side we have the iPhone 7, which we used to get the data from the device, while on the right we have the

Figure 4.7: Example of Real World Test.

Figure 4.8: Mobile application after the test. iPhone 7 on the left and iPhone X on the right.

iPhone X which is the one getting the data from the server. We can see that both of them got the same information which leads us to affirm that the test was successful.

Chapter 5

Conclusions

There are no doubts that we are moving into an ubiquitous world. IoTdevices are becoming cheaper and consequently they are increasing. They are also becoming more and more powerful which allows us to do things today that were impossible a few years ago. The majority of these IoTdevices are made to do simple tasks and communicate through the internet, so they are a really good fit for home automation, collection of meteorological information, etc. With these devices users nowadays tend to expect that the information that they are visualizing is in real time, and this aspect makes a lot of difference for the user perspective about the systems.

Nowadays, in ocean exploration, the costs are extremely high which makes this exploration very slow and also only available to groups with lots of resources. On the contrary, in land, we have many people that contribute for example in animal researches, and they do that as a hobby with money from their own pocket. Most of these people that actively contribute to science can't even imagine to do the same in the ocean mostly because of the costs involved, however this whole situation can change with the adaptation of these cheap IoTdevices.

Our solution tries to pick on one of these IoTdevices and use it as an ocean monitoring sensor, and at the same time tries to provide to the users the data as close to real time as possible.

In order to demonstrate the capabilities of our solution we conducted some performance tests on both the server and the device. We also simulated a real world scenario to prove that our solution can in fact be used in the real world.

5.1 Achievements

The biggest achievement of our solution was the fact that we helped to push the boundaries of what these kind devices are made and capable of doing.

From our requirements set in section 3.1 we actually achieved the majority of them.

Affordable There is no doubt that our system and in particular the device is cheaper that everything else that we have seen so far. The cost of the photon device used in this project is around 20 dollars.

If we combine the costs of all the sensors used in this project we get a cost of 650 euros. Our solution also makes possible to use any kind of sensors that are compatible with an arduino, so the actual cost of the system can change a lot based on the price of the sensors. This enables the system to be used either by groups with lots of money or by people that simply want to explore the ocean as an hobby.

Close to Real Time Data This system allows the users to gather the information in real time if they are close to a device, and if not they will have a greater probability to see the data closer to real time, than in other systems, where the devices needs to be retrieved and the information upload to the server in order to become available.

Modular system Our devices can accommodate any type of sensors, making only one requirement that they are compatible with any arduino. The ease of sensor switching as well as the device, server and mobile application independence makes these system really easy to implement and use.

Public Access API We wanted to create some sort of an API that would allow us to build every single component of the system independently from the others. While we could make this API closed to external entities we wanted to share the information gathered with anyone and that's why the API is the simplest one we can have which allows anyone to get the information and build any tools with that.

Battery Life Unfortunately we couldn't make achieve this requirement. The photon doesn't allow us to block the power given to any sensor, so while the device battery consumption is actually not that big, with the amount of sensors that we were using, the 10 000 mAh powerbank only lasted for about 23 hours.

5.2 Future Work

Since these devices aren't made for this kind of task, we already expected that our solution would be far from perfect. In fact there are a few number of suggestions that may or may not help these kind of systems to become available to almost everyone.

- **Access Point mode photon**
  Particle is currently developing the version 8 of there firmware where they are supposed to enable tcp connections while the device is in listening mode. With this new option, the restrictions that we are currently facing in the limited size of the responses will stop to exist, which will make possible, for example, to send all the sound recordings to the mobile devices. These change will need a complete redesign of the communication mechanism between the device and the mobile application, but without a doubt, this is an huge improvement to the whole system.
- **LoRaWANSupport**
  In chapter 2 we talked about a new technology that supports long range Wi-Fi communication.

While this won't be a solution for monitoring in the whole ocean since the maximum distance nowadays is 20 Km, this solution can be implemented in some places like between Madeira and Porto Santo which have a distance between them of about 40 km. With one antenna in both islands pointing at each other, we could cover almost the entire area. This is also a good solution if we want to monitor the continental shelf around one particular location.

- **Powerful devices**

  Technology is evolving at a rate never seen before. Six years ago maybe no one would think about having IoTdevices in their homes, and nowadays is starting to become a trend. This evolution creates unique opportunities. What were impossible today might actually be easy in a few years, so there is an opportunity to revisit this topic and try to implement it with other devices that might become available in the future.

- **Improved Mobile Application User Interface**

  The main purpose of this solution wasn't the mobile application. But since we developed that we created one unique visualization that wasn't available for the iOS platform. We also had concerns about the interface an usability of the application during the development phase, since we are targeting end-users. One of the key tests that all user interfaces should do is usability tests. However, we haven't done them, so the in future works there is a real possibility to develop some new and more appealing interfaces and properly test them. There is also a possibility to develop the mobile application for multiple platforms, since we only developed for iOS ecosystem.

Bibliography

[1] M. Weiser. Some computer science issues in ubiquitous computing. *Commun. ACM*, 36(7):75–84, July 1993. ISSN 0001-0782. doi: 10.1145/159544.159617. URL http://doi.acm.org/10.1145/159544.159617.

[2] M. Radeta, N. J. Nunes, D. Vasconcelos, and V. Nisi. Poseidon - passive-acoustic ocean sensor for entertainment and interactive data-gathering in opportunistic nautical-activities. In *Proceedings of the 2018 Designing Interactive Systems Conference*, DIS '18, pages 999–1011, New York, NY, USA, 2018. ACM. ISBN 978-1-4503-5198-0. doi: 10.1145/3196709.3196752. URL http://doi.acm.org/10.1145/3196709.3196752.

[3] R. Ashcroft. They walk among us the rise of citizen science, Aug 2016. URL https://www.the-ies.org/sites/default/files/journals/es_citizen_science_aug_16.pdf.

[4] D. J. Trumbull, R. Bonney, D. Bascom, and A. Cabral. Thinking scientifically during participation in a citizen-science project. *Science Education*, 84(2):265–275. doi: 10.1002/(SICI)1098-237X(200003)84:2⟨265::AID-SCE7⟩3.0.CO;2-5. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/%28SICI%291098-237X%28200003%2984%3A2%3C265%3A%3AAID-SCE7%3E3.0.CO%3B2-5.

[5] C. Rutz and G. C. Hays. New frontiers in biologging science. *Biology Letters*, 5(3):289–292, 2009. ISSN 1744-9561. doi: 10.1098/rsbl.2009.0089. URL http://rsbl.royalsocietypublishing.org/content/5/3/289.

[6] S. Bograd, B. Block, D. Costa, and B. Godley. Biologging technologies: new tools for conservation. introduction. *Endangered Species Research*, 10:1–7, Mar 2010. doi: 10.3354/esr00269.

[7] Scor-tagging. URL http://www.coml.org/comlfiles/scor/SCOR-tagging.pdf.

[8] S. J. Cooke, J. D. Midwood, J. D. Thiem, P. Klimley, M. C. Lucas, E. B. Thorstad, J. Eiler, C. Holbrook, and B. C. Ebner. Tracking animals in freshwater with electronic tags: past, present and future. *Animal Biotelemetry*, 1(1):1–19, May 2013. doi: 10.1186/2050-3385-1-5.

[9] D. W. Fuller, K. M. Schaefer, J. Hampton, S. Caillot, B. M. Leroy, and D. G. Itano. Vertical movements, behavior, and habitat of bigeye tuna (thunnus obesus) in the equatorial central pacific ocean. *Fisheries Research*, 172(Supplement C):57 – 70, 2015. ISSN 0165-7836. doi: https://

doi.org/10.1016/j.fishres.2015.06.024. URL http://www.sciencedirect.com/science/article/pii/S0165783615300047.

[10] Worldwide tracking and environmental monitoring by satellite. URL http://www.argos-system.org/.

[11] J. Goldbogen, D. Cade, A. Boersma, J. Calambokidis, S. Kahane-Rapport, P. Segre, A. Stimpert, and A. Friedlaender. Using digital tags with integrated video and inertial sensors to study moving morphology and associated function in large aquatic vertebrates. *The Anatomical Record*, 300(11): 1935–1941, 2017. ISSN 1932-8494. doi: 10.1002/ar.23650. URL http://dx.doi.org/10.1002/ar.23650.

[12] Teach, learn, and make with raspberry pi. URL https://www.raspberrypi.org/.

[13] Daniel. Raspiwhale: First raspberry pi mounted on a whale, Nov 2015. URL http://ideaspi.com/raspiwhale/first-raspberrypi-mounted-on-a-whale/.

[14] M. Greer. Iot development board comparison, Apr 2016. URL https://www.allaboutcircuits.com/news/iot-development-board-comparison/.

[15] D. G. vasconcelos. *Biodiversity Monitoring Using Smart Acoustic Sensors Application to mosquitos*. PhD thesis, 2017.

[16] Particle, . URL https://docs.particle.io/datasheets/photon-(wifi)/photon-datasheet/.

[17] URL https://store.arduino.cc/arduino-uno-rev3.

[18] P. Foundation. Processing.org, . URL https://processing.org/.

[19] Y. Tawil. Understanding arduino uno hardware design, Jul 2016. URL https://www.allaboutcircuits.com/technical-articles/understanding-arduino-uno-hardware-design/.

[20] Adafruit feather huzzah esp8266. URL https://learn.adafruit.com/adafruit-feather-huzzah-esp8266/overview.

[21] Ifttt. URL https://ifttt.com/wtf.

[22] Particle, . URL https://docs.particle.io/guide/tools-and-features/ifttt/.

[23] J. Petäjäjärvi, K. Mikhaylov, M. Pettissalo, J. Janhunen, and J. Iinatti. Performance of a low-power wide-area network based on lora technology: Doppler robustness, scalability, and coverage. *International Journal of Distributed Sensor Networks*, 13(3):155014771769941, 2017. doi: 10.1177/1550147717699412.

[24] Sigfox. URL https://www.sigfox.com/en.

[25] G. Schatz. Sigfox vs. lora: A comparison between technologies and business models. URL https://www.link-labs.com/blog/sigfox-vs-lora.

[26] R. M. Ribeiro. Sistema de monitorização do consumo de água utilizando tecnologia sigfox. Jul 2017. URL http://hdl.handle.net/10216/106909.

[27] Sigfox core service, Nov 2016. URL https://www.youtube.com/watch?v=6ZBGDtmDGRU.

[28] LaureneGroot. Lora alliance introduction, Jul 2015. URL https://www.youtube.com/watch?v=2Y0bMX3TVi0.

[29] lora-alliance. URL https://www.lora-alliance.org/.

[30] Weightless - setting the standard for iot. URL http://www.weightless.org/.

[31] B. Ray. What is weightless? URL https://www.link-labs.com/blog/what-is-weightless.

[32] N. Foundation, . URL https://nodejs.org/en/.

[33] Mongodb for giant ideas. URL https://www.mongodb.com/.

[34] Particle, . URL https://docs.particle.io/reference/firmware/photon/#softap-http-pages.

[35] URL https://expressjs.com/.

[36] C. D. Team. Cocoapods.org. URL https://cocoapods.org/.

[37] Uipageviewcontroller. URL https://developer.apple.com/documentation/uikit/uipageviewcontroller.

[38] Realm: Create reactive mobile apps in a fraction of the time. URL https://realm.io/.

[39] URL https://www.sqlite.org/index.html.

[40] Core data programming guide: What is core data?, Mar 2017. URL https://developer.apple.com/library/archive/documentation/Cocoa/Conceptual/CoreData/index.html.

[41] Bkobkobko. bkobkobko/sparktime. URL https://github.com/bkobkobko/sparktime.

[42] Date and time format - iso 8601, Mar 2018. URL https://www.iso.org/iso-8601-date-and-time-format.html.

[43] Cocoa core competencies, Apr 2018. URL https://developer.apple.com/library/archive/documentation/General/Conceptual/DevPedia-CocoaCore/MVC.html.

[44] Alamofire. Alamofire/alamofire, Aug 2018. URL https://github.com/Alamofire/Alamofire.

[45] Nehotspotconfiguration. URL https://developer.apple.com/documentation/networkextension/nehotspotconfiguration.

[46] Nightingale's rose. URL http://mbostock.github.io/protovis/ex/crimea-rose.html.

[47] Uikit. URL https://developer.apple.com/documentation/uikit.

[48] Uiscrollview. URL https://developer.apple.com/documentation/uikit/uiscrollview.

[49] D. Gindi. danielgindi/charts, Sep 2018. URL https://github.com/danielgindi/Charts.

[50] Mapkit. URL https://developer.apple.com/documentation/mapkit.

[51] 1150 protector case. URL https://peliproducts.co.uk/1150-protector-case.html.

[52] Apache jmeter™. URL http://jmeter.apache.org/.

Appendix A

Technical Datasheets

A.1 Particle Photon Project

```
//------- Librarys Imports
#include "Particle.h"
#include "SdFat.h"
#include "softap_http.h"
#include "math.h"
#include "Base64RK.h"


//------- My Imports
#include "config.h"

#include "sensors/Bar100Sensor.h"
#include "sensors/TurbiditySensor.h"
#include "sensors/PHMeterSensor.h"
#include "sensors/GPSSensor.h"
#include "sensors/DissolvedOxygenSensor.h"
#include "sensors/TimeManager.h"

#include "sound/SoundManager.h"

int BUFFER_SIZE = 5120;

Config cfg = getDefaultConfig();

//------- Needed sensors ----- Don't remove them
GPSSensor* gpsSensor;
TimeManager* timeManager;


//------- My Sensors Objects // Add New Sensors here
Bar100Sensor* bar100Sensor;
TurbiditySensor* turbiditySensor;
PHMeterSensor* phMeterSensor;
DissolvedOxygenSensor* dissolvedOxygenSensor;

int lock = 0;


//ENABLES threads needed for softap Mode
SYSTEM_THREAD(ENABLED)

//------- SDCARD SETTINGS
SdFatSoftSpi<D3, D2, D4> sd;
const uint8_t chipSelect = D5;
File myFile;

bool canRecord = false;
bool softAPMode = false;

// recordingSoundState
enum State { STATE_STARTRECORDING, STATE_RUNNING, STATE_FINISH};
State state = STATE_STARTRECORDING;
unsigned long recordingStart;

// Forward Declaration of functions
void myPages(const char* url, ResponseCallback* cb, void* cbArg, Reader*
 body, Writer* result, void* reserved);


// setup() runs once, when the device is first turned on.
void setup() {
  setupSerial(); //allows output in serial Monitor
  waitUntil(Particle.connected); // waitUntil network is available
  Serial.println("Connected to Particle ");



  // register the cloud function
  Particle.function("enableap", enableSoftAp); // function to start AP Mode
  Particle.function("enableread", enableReading); // function to start
   recording Mode

  // Needed Objects
  gpsSensor = new GPSSensor();
  timeManager = new TimeManager();

  //Initialize your Objects
  bar100Sensor = new Bar100Sensor(997); // kg/m^3 (freshwater, 1029 for
   seawater)
  turbiditySensor = new TurbiditySensor();
  phMeterSensor = new PHMeterSensor();
  dissolvedOxygenSensor = new DissolvedOxygenSensor();

}

// loop() runs over and over again, as quickly as it can execute.
void loop() {
  if(canRecord){
    recordingSoundLoop();
  }
  // checks if the current depth is less than 1 meter, if it is
  // disables the reading mode and turns the softAP mode on
  if(softAPMode == true){

  }

  // checks if current depth is bellow 1 meter and if softAP is on, if it is
  // disables the softAP mode and turns the reading mode on
  /*if(bar100Sensor->getDepth() < -1 && softAPMode == true){
    canRecord = true;

    softAPMode = false;

    WiFi.listen(false); // Disables Listening Mode
  }*/
}


// Reads the sensors values
void readSensors(){
```

```
  timeManager->record();

  gpsSensor->record();

  bar100Sensor->record();

  turbiditySensor->record();

  phMeterSensor->record();

  dissolvedOxygenSensor->setTemperature(bar100Sensor->getTemperature());
  dissolvedOxygenSensor->record();

  // dissolvedOxygenSensor needs 10 values before reading is correct
  if(dissolvedOxygenSensor->getRecordValue() == "0.00,"){
    delay(100);
    readSensors();
  }

}

// print result of sensors for debugging
void printResults(){
  Serial.println("Date:");
  Serial.println(timeManager->getRecordValue());

  Serial.println("-----------------");
  Serial.println("-----------------");

  Serial.println("GPS    Lat, Long");
  Serial.println(gpsSensor->getRecordValue());


  Serial.println("-----------------");
  Serial.println("-----------------");

  Serial.println("temperature, depth, altitude, pressure");
  Serial.println(bar100Sensor->getRecordValue());

  Serial.println("-----------------");
  Serial.println("-----------------");

  Serial.println("Turbidity  1 == HIGH || 0 == LOW");
  Serial.println(turbiditySensor->getRecordValue());

  Serial.println("-----------------");
  Serial.println("-----------------");

  Serial.println("PH: ");
  Serial.println(phMeterSensor->getRecordValue());

  Serial.println("-----------------");
  Serial.println("-----------------");

  Serial.println("DissolvedOxygenSensor: ");
  Serial.println(dissolvedOxygenSensor->getRecordValue());

  Serial.println("-----------------");
  Serial.println("-----------------");

  String dataToWrite = timeManager->getRecordValue() + ","; //date
  dataToWrite += gpsSensor->getRecordValue() + ",";         //gps ->
   latitude,longitude
  dataToWrite += bar100Sensor->getRecordValue() + ",";      //
   temperature,depth,altitude,pressure
  dataToWrite += turbiditySensor->getRecordValue() + ",";   //turbidity
  dataToWrite += phMeterSensor->getRecordValue() + ",";     //ph
  dataToWrite += dissolvedOxygenSensor->getRecordValue();   //oxygen

  Serial.println(dataToWrite);
}

// recordingSoundLoop in the end of the loop calls the timeManager to check
 if we can read the sensors
void recordingSoundLoop(){
  switch (state) {
    // In this state we can start record
    case STATE_STARTRECORDING:
      connectSDCard();
      changeDirectory("Audio");
      startRecordingState(getAudiFileName());
      recordingStart = millis();
      state = STATE_RUNNING;
      break;

    // In this state we are waiting for the microphone record to finish
    case STATE_RUNNING:
      // check to see if if the recording time exceeded the maximum time
      if (millis() - recordingStart >= cfg.MAX_RECORDING_LENGTH_MS) {
        state = STATE_FINISH;
      } else {
        recordingState();
      }
      break;

    // In this state we save all the sensors information to the SdCard
    // Then setDelay for the next reading to start
    case STATE_FINISH:
      finishState();
      changeDirectory("/");
      state = STATE_STARTRECORDING;

      if(timeManager->canReadSensors()){
        readSensors();
        saveRecordedData();
      }
      delay(1000);

      break;
  }
```

```
}

// Current Data
//"date,audioFile,latitude,longitude,temperature,depth,altitude,pressure,tur
 bidity,ph,oxygen"
void saveRecordedData(){
  if (!sd.exists(cfg.fileName)) {
    Serial.println("File don't exists creating it now: " + cfg.fileName);
    // open the file for write at end like the "Native SD library"
    if (!myFile.open(cfg.fileName, O_RDWR | O_CREAT | O_AT_END)) {
      sd.errorHalt(("opening " + cfg.fileName + " for write failed"));
    }
    myFile.println(cfg.csvInitialString);
  }
  else {
    if (!myFile.open(cfg.fileName, O_RDWR | O_CREAT | O_AT_END)) {
      sd.errorHalt(("opening " + cfg.fileName + " for write failed"));
    }
  }
  String dataToWrite = timeManager->getRecordValue() + ","; //date
  dataToWrite += gpsSensor->getRecordValue() + ",";         //gps ->
   latitude,longitude
  dataToWrite += bar100Sensor->getRecordValue() + ",";      //
   temperature,depth,altitude,pressure
  dataToWrite += turbiditySensor->getRecordValue() + ",";   //turbidity
  dataToWrite += phMeterSensor->getRecordValue() + ",";     //ph
  dataToWrite += dissolvedOxygenSensor->getRecordValue();   //oxygen

  Serial.println(dataToWrite);
  myFile.println(dataToWrite);
  myFile.close();

}

// gets a name for the audioFile based on the Currentdate
String getAudiFileName(){
  String aux = timeManager->getDateNow().replace(":", "");
  return cfg.DEVICE_ID + aux + ".wav";
}


// Sets the serial to enable debugging
void setupSerial(){
  //---------------------- Serial setup
  Serial.begin(9600);
  // Wait for USB Serial
  while (!Serial) {
    SysCall::yield();
  }
}

// connects the sdCard
void connectSDCard(){
  // Initialize SdFat or print a detailed error message and halt
  // Use half speed like the native library.
  // Change to SPI_FULL_SPEED for more performance.
  if (!sd.begin(chipSelect, SPI_HALF_SPEED)) {
    sd.initErrorHalt();
  }
}

// changes directory inside the SD card
void changeDirectory(String path){
  if(!sd.chdir(path, true)){
    Serial.println("error changing directory: "+ path);
    sd.errorHalt("error changing directory");
  }
}

// function than handles the requests to the device while in softap Mode
void myPage(const char* url, ResponseCallback* cb, void* cbArg, Reader*
 body, Writer* result, void* reserved)
{

      Serial.printlnf("handling page %s", url);

      if (strcmp(url,"/index")==0) {
          Serial.println("sending redirect");
          Header h("Location: /index.html\r\n");
          cb(cbArg, 0, 301, "text/plain", &h);
          return;
      }

      String auxURl = url;
      String dataPart = auxURl.substring(auxURl.indexOf("?") + 1,
       auxURl.length());
      Serial.println("dataPat: " + dataPart);

      if (strcmp(auxURl.substring(0,auxURl.indexOf("?")),"/data")==0) {
          cb(cbArg, 0, 200, "text/json", nullptr);
          result->write(dataToSend(dataPart));
      }
      else {
          cb(cbArg, 0, 404, nullptr, nullptr);
      }
}

// generates a string with the response to the client,
// based on the last part of the data that the client received
String dataToSend(String dataPart){
    connectSDCard();
    int i;
    if (dataPart.length() == 0) {
      Serial.println("sending new data");
      i = 0;
    }
    else {
      i = dataPart.toInt();
      Serial.println("sending data from: " + dataPart);
    }
```

```
    String response = "";

    WITH_LOCK(Serial){
      // open the file to read
      if (!myFile.open(cfg.fileName, O_READ)) {
        sd.errorHalt(("opening  " + cfg.fileName + " for read failed"));
      }

      // read from the file until there's nothing else in it:
      int data;


      myFile.seek(i);

      while ((data = myFile.read()) >= 0) {

        char c = data;
        if (c == '\n' || c == '\r') { // substitutes \n and \r to ; so that
         json can be accepted
          if (!response.endsWith(";")) {
            response += ";";
          }
        } else {
          response += c;
        }
        i ++;

        if (response.length() >= BUFFER_SIZE) {
          myFile.close();
          return getInitialJsonBody(i) + response + getFinalJsonbody(false);
        }
      }
      myFile.close();
      Serial.println("done");
    }
    return getInitialJsonBody(i) + response + getFinalJsonbody(true);;
}

// generates the initial jsonBody
String getInitialJsonBody(int dataPart){
  String jsonBody = "{ \"deviceId\": \"";
  jsonBody += cfg.DEVICE_ID;
  jsonBody += "\",\"dataPart\":";
  jsonBody += dataPart;
  jsonBody += ",\"csvData\":\"";
  return jsonBody;
}

// generates the final jsonBody
String getFinalJsonbody(bool b){
  String jsonBody = "\",\"isFinalData\":";
  if (b) {
    jsonBody += "true";
  } else {
    jsonBody += "false";
  }

  jsonBody += "}";
  return jsonBody;
}

// Debug functions
int enableSoftAp(String s){
  //starts listening Mode with a timout
  if (canRecord) {
    canRecord = false;
  }
  WiFi.listen();
  softAPMode = true;
  softap_set_application_page_handler(myPage, nullptr);
  Serial.println("Now Listening: ");
  return 0;
}



// Debug functions
int enableReading(String s){
  canRecord = !canRecord;
  if (canRecord) {
    return 0;
  }
  return 1;
}
#include "Particle.h"
#include "SensorsInterface.h"
#include "KellerLD.h"

// Implementation of the Bar100Sensor
class Bar100Sensor: public SensorsInterface {

  // Fluid density should be changed if we are dealing with salt water
  int _fluidDensity;

  // object of the creators' library
  KellerLD sensor;

  public:
    // constructor class where the fluidDensity is set
    Bar100Sensor(int fluidDensity);

    // makes the sensor record the values
    virtual void record();

    // returns all the values as a string
    virtual String getRecordValue();

    // returns the current temperature value that's needed by other sensors
    float getTemperature();
```

```
    // returns the current depth value that's needed by the main loop
    float getDepth();
};
#include "SensorsInterface.h"

// Turbidity Sensor
class TurbiditySensor: public SensorsInterface {

  //------- PIN
  int turbiditySensorPin = D5;  // Pin where the sensor is connected
  int turbidity;  // turbidity value

  public:
    TurbiditySensor();  // object constructor
    virtual void record();  // record the current turbidity value
    virtual String getRecordValue();  // returns the turbidity value
     recorded as a string
};
#include "SensorsInterface.h"

// pH Meter sensor
class PHMeterSensor: public SensorsInterface {

  int phSensorPin = A2; // Pin where the sensor is connected
  float offset = 0.35;  // offset value chosen after calibration
  float ph; // current pH value

  public:
    PHMeterSensor();   // object constructor
    virtual void record();  // record the current pH value
    virtual String getRecordValue();  // returns the pH value recorded as a
     string
};
#include "SensorsInterface.h"
#include "avr/pgmspace.h"

#define VREF 5000    //for arduino uno, the ADC reference is the AVCC, that
 is 5000mV(TYP)

#define EEPROM_write(address, p) {int i = 0; byte *pp = (byte*)&(p);for(; i
 < sizeof(p); i++) EEPROM.write(address+i, pp[i]);}
#define EEPROM_read(address, p)  {int i = 0; byte *pp = (byte*)&(p);for(; i
 < sizeof(p); i++) pp[i]=EEPROM.read(address+i);}

#define ReceivedBufferLength 20
#define SCOUNT  30           // sum of sample point

#define SaturationDoVoltageAddress 12          //the address of the
 Saturation Oxygen voltage stored in the EEPROM
#define SaturationDoTemperatureAddress 16      //the address of the
 Saturation Oxygen temperature stored in the EEPROM

// implementation of the DissolvedOxygenSensor code from manufacturers
// The calibration needs to be done prior the first usage
// https://www.dfrobot.com/wiki/index.php/
 Gravity:_Analog_Dissolved_Oxygen_Sensor_SKU:SEN0237
class DissolvedOxygenSensor: public SensorsInterface {
  int DoSensorPin = A5; // Pin where the sensor is connected
  float doValue;        // current dissolved oxygen value, unit; mg/L
  float temperature;    // value of the current temperature

  char receivedBuffer[ReceivedBufferLength+1];    // store the serial
   command
  byte receivedBufferIndex = 0;

  int analogBuffer[SCOUNT];    //store the analog value in the array, readed
   from ADC
  int analogBufferTemp[SCOUNT];
  int analogBufferIndex = 0,copyIndex = 0;

  float SaturationDoVoltage,SaturationDoTemperature;
  float averageVoltage;

  const float SaturationValueTab[41] PROGMEM = {      //saturation dissolved
   oxygen concentrations at various temperatures
    14.46, 14.22, 13.82, 13.44, 13.09,
    12.74, 12.42, 12.11, 11.81, 11.53,
    11.26, 11.01, 10.77, 10.53, 10.30,
    10.08, 9.86,  9.66,  9.46,  9.27,
    9.08,  8.90,  8.73,  8.57,  8.41,
    8.25,  8.11,  7.96,  7.82,  7.69,
    7.56,  7.43,  7.30,  7.18,  7.07,
    6.95,  6.84,  6.73,  6.63,  6.53,
    6.41,
    };

  public:
    DissolvedOxygenSensor();  // initializes the sensor
    virtual void record();    // records the value and saves it
    virtual String getRecordValue();  // returns the value recorded as a
     String

    void setTemperature(float temp);  // updates the current temperature

  private:  // Private methods mainly used for calibration
    boolean serialDataAvailable(void);
    byte uartParse();
    void doCalibration(byte mode);
    int getMedianNum(int bArray[], int iFilterLen);
    void readDoCharacteristicValues(void);
};
#include "SensorsInterface.h"
#include "SparkTime/SparkTime.h"
#include "../config.h"


// Class that manages the time
// Uses the SparkTime library and needs internet connection from the device
 in the
```

```
// beginning in order to get the current Time
class TimeManager: public SensorsInterface {

  UDP UDPClient;  // UDP client to connect to the server
  SparkTime rtc;  // RTC object

  unsigned long currentTime;
  unsigned long lastTime;
  String timeStr;

  public:
    TimeManager();  // initializes the rtc
    virtual void record(); // record the value from the sensor
    virtual String getRecordValue(); // returns the date in string format
    String getDateNow();  // gets the current date
    bool canReadSensors();  // checks if it's possible to read the current
     time

};
#include "Particle.h"

#ifndef SENSORINTERFACE_H
#define SENSORINTERFACE_H

// The code for each sensor object must be implemented by complying with
 this interface
// So every sensor object must implement this interface and their respective
 Methods.
class SensorsInterface  {

  public:
    // record the value from the sensor for later use
    virtual void record() = 0;

    // gets the value from the sensor as a string.
    // If you want to retur more than one value for a specific sensor they
     should be
    // separed by a comma
    virtual String getRecordValue() = 0;
};

#endif
#include "SensorsInterface.h"
#include "TinyGPS/TinyGPS.h"


// Gps sensor using the TinyGPS Library
class GPSSensor: public SensorsInterface {

  TinyGPS gps;
  char szInfo[64];

  public:
    GPSSensor();  // initializes the sensor
    virtual void record();  // records the value and saves it
    virtual String getRecordValue();  // returns the value recorded as a
     String
};
#ifndef MYSTRUCT_H
#define MYSTRUCT_H

struct Config {
  // Device configs
  String DEVICE_ID; // Id of Device should be unique for each photon
  int SDCARD_PIN;   // Pin for the sd card
  String fileName; // Name of the file to save the data
  String csvInitialString; // Initial String of csvData change this to
   conform sensors available
  int TIME_ZONE;   // Timezone 0 for gmt then + or - for each zone

    int BUFFER_SIZE; // Maximum Size of the buffer to send data


    // recording vars
    unsigned long MAX_RECORDING_LENGTH_MS;  // Maximum Recording Time
  long SAMPLE_RATE;                       // Audio Sample Rate -> Minimum 8k
   maximum 48k because of timer overhead
  int MICRO_PIN;                          // Microphone PIN

  ushort sampleRate;
  ushort sampleBufferSize;
  ushort bytesPerSec;
  size_t dataSize;
  size_t fileSize;
};

Config getDefaultConfig();

#endif
#include "SdFat.h"
#include "Particle.h"
#include "../config.h"
#include "application.h"

#include "SparkIntervalTimer.h"


#ifndef SOUND_MANAGER_H
#define SOUND_MANAGER_H


//----- FUNCTIONS
void writeWavHeader(String fileName); // writes the header at the beginning
 of the .wav file
void writeOutHeader();  // writes the out header at the end of the .wav file
void timerISR(); // timer function to read the data from the microphone

void startRecordingState(String fileName);  // function that manages the
 state STATE_STARTRECORDING
void recordingState();  // function that manages the state STATE_RUNNING
```

```
void finishState();   // function that manages the state STATE_FINISH


#endif
#include "GPSSensor.h"


// Public Methods Defenition
GPSSensor::GPSSensor(){
  Serial1.begin(9600);
  sprintf(szInfo, "0.0,0.0");
}

// records the current location if we have valid gps coordinates, if not,
// we don't change the current location
void GPSSensor::record(){
  bool isValidGPS = false;

    for (unsigned long start = millis(); millis() - start < 1000;){
        // Check GPS data is available
        while (Serial1.available()){
            char c = Serial1.read();
            // parse GPS data
            if (gps.encode(c))
                isValidGPS = true;
        }
    }

    // If we have a valid GPS location then publish it
    if (isValidGPS){
        float lat, lon;
        unsigned long age;

        gps.f_get_position(&lat, &lon, &age);

        sprintf(szInfo, "%.6f,%.6f", (lat == TinyGPS::GPS_INVALID_F_ANGLE ?
         0.0 : lat), (lon == TinyGPS::GPS_INVALID_F_ANGLE ? 0.0 : lon));
    }
}

// returns the value recorded as a String
String GPSSensor::getRecordValue(){
  String str = szInfo;
  return str;
}
#include "DissolvedOxygenSensor.h"


// Public Methods Defenition
DissolvedOxygenSensor::DissolvedOxygenSensor(){
  pinMode(DoSensorPin,INPUT);
  readDoCharacteristicValues();      //read Characteristic Values calibrated
   from the EEPROM
}

void DissolvedOxygenSensor::record(){
  static unsigned long analogSampleTimepoint = millis();
   if(millis()-analogSampleTimepoint > 30U)     //every 30 milliseconds,read
    the analog value from the ADC
   {
     analogSampleTimepoint = millis();
     analogBuffer[analogBufferIndex] = analogRead(DoSensorPin);    //read
      the analog value and store into the buffer
     analogBufferIndex++;
     if(analogBufferIndex == SCOUNT)
         analogBufferIndex = 0;
   }

   static unsigned long printTimepoint = millis();
   if(millis()-printTimepoint > 1000U)
   {
      printTimepoint = millis();
      for(copyIndex=0;copyIndex<SCOUNT;copyIndex++)
      {
        analogBufferTemp[copyIndex]= analogBuffer[copyIndex];
      }
      averageVoltage = getMedianNum(analogBufferTemp,SCOUNT) * (float)VREF /
       1024.0; // read the value more stable by the median filtering
       algorithm
      //Serial.print(F("Temperature:"));
      //Serial.print(temperature,1);
      //Serial.print(F("^C"));
      doValue = pgm_read_float_near( &SaturationValueTab[0] + (int)
       (SaturationDoTemperature+0.5) ) * averageVoltage /
       SaturationDoVoltage;  //calculate the do value, doValue = Voltage /
       SaturationDoVoltage * SaturationDoValue(with temperature
       compensation)
      //Serial.print(F(",  DO Value:"));
      //Serial.print(doValue,2);
      //Serial.println(F("mg/L"));
   }

   if(serialDataAvailable() > 0)
   {
      byte modeIndex = uartParse();  //parse the uart command received
      doCalibration(modeIndex);    // If the correct calibration command is
       received, the calibration function should be called.
   }

}

String DissolvedOxygenSensor::getRecordValue(){
  return String(doValue,2) + ",";
}

void DissolvedOxygenSensor::setTemperature(float temp){
  temperature = temp;
}
```

```
boolean DissolvedOxygenSensor::serialDataAvailable(void){
  char receivedChar;
  static unsigned long receivedTimeOut = millis();
  while ( Serial.available() > 0 )
  {
    if (millis() - receivedTimeOut > 500U)
    {
      receivedBufferIndex = 0;
      memset(receivedBuffer,0,(ReceivedBufferLength+1));
    }
    receivedTimeOut = millis();
    receivedChar = Serial.read();
    if (receivedChar == '\n' || receivedBufferIndex == ReceivedBufferLength)
    {
    receivedBufferIndex = 0;
    strupr(receivedBuffer);
    return true;
    }else{
        receivedBuffer[receivedBufferIndex] = receivedChar;
        receivedBufferIndex++;
    }
  }
  return false;
}

byte DissolvedOxygenSensor::uartParse(){
    byte modeIndex = 0;
    if(strstr(receivedBuffer, "CALIBRATION") != NULL)
        modeIndex = 1;
    else if(strstr(receivedBuffer, "EXIT") != NULL)
        modeIndex = 3;
    else if(strstr(receivedBuffer, "SATCAL") != NULL)
        modeIndex = 2;
    return modeIndex;
}

void DissolvedOxygenSensor::doCalibration(byte mode){
    char *receivedBufferPtr;
    static boolean doCalibrationFinishFlag = 0,enterCalibrationFlag = 0;
    float voltageValueStore;
    switch(mode)
    {
      case 0:
      if(enterCalibrationFlag)
         Serial.println(F("Command Error"));
      break;

      case 1:
      enterCalibrationFlag = 1;
      doCalibrationFinishFlag = 0;
      Serial.println();
      Serial.println(F(">>>Enter Calibration Mode<<<"));
      Serial.println(F(">>>Please put the probe into the saturation oxygen
       water! <<<"));
      Serial.println();
      break;

     case 2:
      if(enterCalibrationFlag)
      {
         Serial.println();
         Serial.println(F(">>>Saturation Calibration Finish!<<<"));
         Serial.println();
         EEPROM_write(SaturationDoVoltageAddress, averageVoltage);
         EEPROM_write(SaturationDoTemperatureAddress, temperature);
         SaturationDoVoltage = averageVoltage;
         SaturationDoTemperature = temperature;
         doCalibrationFinishFlag = 1;
      }
      break;

        case 3:
        if(enterCalibrationFlag)
        {
            Serial.println();
            if(doCalibrationFinishFlag)
               Serial.print(F(">>>Calibration Successful"));
            else
              Serial.print(F(">>>Calibration Failed"));
            Serial.println(F(",Exit Calibration Mode<<<"));
            Serial.println();
            doCalibrationFinishFlag = 0;
            enterCalibrationFlag = 0;
        }
        break;
    }
}

int DissolvedOxygenSensor::getMedianNum(int bArray[], int iFilterLen) {
      int bTab[iFilterLen];
      for (byte i = 0; i<iFilterLen; i++)
      {
      bTab[i] = bArray[i];
      }
      int i, j, bTemp;
      for (j = 0; j < iFilterLen - 1; j++)
      {
      for (i = 0; i < iFilterLen - j - 1; i++)
          {
        if (bTab[i] > bTab[i + 1])
            {
        bTemp = bTab[i];
            bTab[i] = bTab[i + 1];
        bTab[i + 1] = bTemp;
         }
      }
      }
      if ((iFilterLen & 1) > 0)
    bTemp = bTab[(iFilterLen - 1) / 2];
      else
```

```
    bTemp = (bTab[iFilterLen / 2] + bTab[iFilterLen / 2 - 1]) / 2;
      return bTemp;
}

void DissolvedOxygenSensor::readDoCharacteristicValues(void){
    EEPROM_read(SaturationDoVoltageAddress, SaturationDoVoltage);
    EEPROM_read(SaturationDoTemperatureAddress, SaturationDoTemperature);
    if(EEPROM.read(SaturationDoVoltageAddress)==0xFF &&
     EEPROM.read(SaturationDoVoltageAddress+1)==0xFF &&
     EEPROM.read(SaturationDoVoltageAddress+2)==0xFF &&
     EEPROM.read(SaturationDoVoltageAddress+3)==0xFF)
    {
      SaturationDoVoltage = 1127.6;   //default voltage:1127.6mv
      EEPROM_write(SaturationDoVoltageAddress, SaturationDoVoltage);
    }
    if(EEPROM.read(SaturationDoTemperatureAddress)==0xFF &&
     EEPROM.read(SaturationDoTemperatureAddress+1)==0xFF &&
     EEPROM.read(SaturationDoTemperatureAddress+2)==0xFF &&
     EEPROM.read(SaturationDoTemperatureAddress+3)==0xFF)
    {
      SaturationDoTemperature = 25.0;   //default temperature is 25^C
      EEPROM_write(SaturationDoTemperatureAddress, SaturationDoTemperature);
    }
}
#include "TurbiditySensor.h"



// Public Methods Defenition

// Initializes the sensor in the respective pin
TurbiditySensor::TurbiditySensor(){
  pinMode(turbiditySensorPin, INPUT);
}

// reads the value from the pin and saves it
void TurbiditySensor::record(){
  // if 1 == HIGH if 0 == LOW
  turbidity = digitalRead(turbiditySensorPin);
}

// returns the turbidity value that is saved
String TurbiditySensor::getRecordValue(){
  return String(turbidity);
}
#include "Bar100Sensor.h"


// Public Methods Defenition

// constructor method that initializes the sensor and set its fluidDensity
Bar100Sensor::Bar100Sensor(int fluidDensity){
  _fluidDensity = fluidDensity;

  Wire.begin();
  sensor.init();
  sensor.setFluidDensity(fluidDensity);

  if (sensor.isInitialized()) {
    Serial.println("Bar100 Sensor connected.");
  } else {
    Serial.println("Sensor not connected.");
  }
}

// reads all the values from the sensor and saves them in the sensor object
void Bar100Sensor::record(){
  sensor.read();
}

// returns all the values in a string, where each value is separated by a
 comma
String Bar100Sensor::getRecordValue(){
  String str = String(sensor.temperature(),2) + ",";
  str += String(sensor.depth(),2) + ",";
  str += String(sensor.altitude(),2) + ",";
  str += String(sensor.pressure() * 0.001,2); // convert to bar
  return str;
}

// returns the current temperature
float Bar100Sensor::getTemperature(){
  return sensor.temperature();
}

// returns the current Depth
float Bar100Sensor::getDepth(){
  return sensor.depth();
}
#include "PHMeterSensor.h"

// Construction object, doesn't need to do anything in this particular case
PHMeterSensor::PHMeterSensor(){
}

// record the current pH value and saves it
void PHMeterSensor::record(){
  unsigned long int avgValue;  //Store the average value of the sensor
   feedback
  float b;
  int buf[10],temp;

  for(int i=0;i<10;i++){       //Get 10 sample value from the sensor for
   smooth the value
    buf[i]=analogRead(phSensorPin);
    delay(10);
  }
  for(int i=0;i<9;i++){        //sort the analog from small to large
    for(int j=i+1;j<10;j++){
```

```
      if(buf[i]>buf[j]){
        temp=buf[i];
        buf[i]=buf[j];
        buf[j]=temp;
      }
    }
  }
  avgValue=0;
  for(int i=2;i<8;i++)                      //take the average value of 6
   center sample
    avgValue+=buf[i];
  float phValue=(float)avgValue*3.3/4095/6; //convert the analog into
   millivolt
  phValue=3.5*phValue;                      //convert the millivolt into pH
   value
  ph = phValue + offset;
}

// returns the recorded value in a string
String PHMeterSensor::getRecordValue(){
  return String(ph,2);
}
#include "TimeManager.h"


// initializes the rtc in the TIME_ZONE deffined in the config
TimeManager::TimeManager(){
  Config cfg = getDefaultConfig();
  rtc.begin(&UDPClient, "north-america.pool.ntp.org");
  rtc.setTimeZone(cfg.TIME_ZONE);
  lastTime = rtc.hour(rtc.now()) - 1;
}

// updates the currentTime value
void TimeManager::record(){
  currentTime = rtc.now();
}

// returns the value saved in currentTime in ISO format
String TimeManager::getRecordValue(){
  return rtc.ISODateUTCString(currentTime);
}

// returns the actual current time in ISO format
String TimeManager::getDateNow(){
  return rtc.ISODateUTCString(rtc.now());
}

// checks if it's possible to call the sensors for reading
// change this function if you want the sensors to read at a different time
 interval
// currently the sensors will record every hour
bool TimeManager::canReadSensors(){
  if(lastTime != rtc.hour(rtc.now())){
    Serial.println("Time is different");
    lastTime = rtc.hour(rtc.now());
    return true;
  }
  return false;
}
#include "Particle.h"
#include "config.h"

Config getDefaultConfig(){

    Config c = {
        "PhotonTest1",  //Id of Device should be unique for each photon
        12,                         //Pin for the sd card
        "data.csv",     //FileName
        "date,latitude,longitude,temperature,depth,altitude,pressure,turbidi
         ty,ph,oxygen",                         //Initial String of csvData
         change this to conform sensors available
        0,                          // Timezone 0 for gmt then + or - for
         each zone

        5120,                   //BUFFER_SIZE


        // recording vars
        4000,                   // Maximum Recording Time
        16000,                  // Audio Sample Rate
        11,                         // PIN A1

        8000,                       // sampleRate;
        16,                         // sampleBufferSizeKB;
        16000,                  // bytesPerSec = sampleRate * 2
        131072,                 // dataSize = config.DataSize(128 KB) *
         1024; This value MUST ALWAYS be a power of 2
        131116,                 // fileSize = config.dataSize + 44 (Wave
         header)
    };
    return c;
}
#include "SoundManager.h"

File rec;   // object of the file
IntervalTimer timer;    // Object from IntervalTimer library

Config CONFIG = getDefaultConfig(); //config Structure


//------------Variable for wav file-------------------//
#define ADC_CENTER 2060

#define WAVE_CHUNK 16
#define WAVE_TYPE 1
#define NUM_CHANNELS 1
#define BLOCK_ALIGN 2  //numchannel*bitpersample/8
#define BITS_PER_SAMPLE 16
```

```
byte byte1, byte2, byte3, byte4;

//------Structure to hold the data timer-------//
typedef struct {
    volatile bool free = true;
    volatile size_t  index = 0;
    byte data[1024]; // = new byte[24576];
} SampleBuf;

SampleBuf buffers[2]; // = new SampleBuf[CONFIG.numberBuffers];
volatile size_t sampleIndex = 0;
volatile size_t sendIndex = 0;

//----------buffer to send byte for server/TCP-----//
byte buf00[1024];

//----- AUX VARS

int writer = 1;
short sample = 0;
bool firstTime = true;

// function that starts the recording
void startRecordingState(String fileName){
    writeWavHeader(fileName);
    Serial.println("Start Recording audio");
    Serial.println(fileName);

    // Initializes the buffer
    for(size_t ii = 0; ii < 2; ii++) {
                buffers[ii].free = true;
                buffers[ii].index = 0;
            }
            sampleIndex = 0;
            sendIndex = 0;

    // the timer never doesn't need to end after the first call since we are
     always listening
    if (firstTime) {
        timer.begin(timerISR, 1000000 / CONFIG.sampleRate, uSec, TIMER4);
        firstTime = false;
    }
    writer = 0;
}

// the state where we check if the buffer is full, and if it is, we save it
 to the SD card
void recordingState(){
    if (sendIndex < sampleIndex) {
    // There is a sample buffer to send
        SampleBuf *sb = &buffers[sendIndex % 2];
        writer = 1;
        rec.write(sb->data,CONFIG.sampleBufferSize);
        writer = 0;
        sb->free = true;
        sendIndex++;
    }


}

// finish Recording and closing the file
void finishState(){
    Serial.println("Finish Recording");
    writer = 1;
    sampleIndex = 0;
  writeOutHeader();

    sendIndex = 0;
}


// opening a new file and write the initial wav header
void writeWavHeader(String fileName) {

    if (!rec.open(fileName, O_CREAT | O_TRUNC | O_RDWR)) {
        Serial.println("error opening file: " + fileName);
  }

  rec.write("RIFF");
  byte1 = (CONFIG.fileSize-8) & 0xff;
  byte2 = ((CONFIG.fileSize-8) >> 8) & 0xff;
  byte3 = ((CONFIG.fileSize-8) >> 16) & 0xff;
  byte4 = ((CONFIG.fileSize-8) >> 24) & 0xff;
  rec.write(byte1);  rec.write(byte2);  rec.write(byte3);  rec.write(byte4);
  rec.write("WAVE");
  rec.write("fmt ");
  byte1 = WAVE_CHUNK & 0xff;
  byte2 = (WAVE_CHUNK >> 8) & 0xff;
  byte3 = (WAVE_CHUNK >> 16) & 0xff;
  byte4 = (WAVE_CHUNK >> 24) & 0xff;
  rec.write(byte1);  rec.write(byte2);  rec.write(byte3);  rec.write(byte4);
  byte1 = WAVE_TYPE & 0xff;
  byte2 = (WAVE_TYPE >> 8) & 0xff;
  rec.write(byte1);  rec.write(byte2);
  byte1 = NUM_CHANNELS & 0xff;
  byte2 = (NUM_CHANNELS >> 8) & 0xff;
  rec.write(byte1);  rec.write(byte2);
  byte1 = CONFIG.sampleRate & 0xff;
  byte2 = (CONFIG.sampleRate >> 8) & 0xff;
  byte3 = (CONFIG.sampleRate >> 16) & 0xff;
  byte4 = (CONFIG.sampleRate >> 24) & 0xff;
  rec.write(byte1);  rec.write(byte2);  rec.write(byte3);  rec.write(byte4);
  byte1 = CONFIG.bytesPerSec & 0xff;
  byte2 = (CONFIG.bytesPerSec >> 8) & 0xff;
  byte3 = (CONFIG.bytesPerSec >> 16) & 0xff;
  byte4 = (CONFIG.bytesPerSec >> 24) & 0xff;
  rec.write(byte1);  rec.write(byte2);  rec.write(byte3);  rec.write(byte4);
  byte1 = BLOCK_ALIGN & 0xff;
  byte2 = (BLOCK_ALIGN >> 8) & 0xff;
```

```
  rec.write(byte1);  rec.write(byte2);
  byte1 = BITS_PER_SAMPLE & 0xff;
  byte2 = (BITS_PER_SAMPLE >> 8) & 0xff;
  rec.write(byte1);  rec.write(byte2);
  rec.write("data");
  byte1 = CONFIG.dataSize & 0xff;
  byte2 = (CONFIG.dataSize >> 8) & 0xff;
  byte3 = (CONFIG.dataSize >> 16) & 0xff;
  byte4 = (CONFIG.dataSize >> 24) & 0xff;
  rec.write(byte1);  rec.write(byte2);  rec.write(byte3);  rec.write(byte4);
}

// closing the file
void writeOutHeader(){
    rec.close();
}

// timer function where it saves the recording bytes to a buffer, it's
 called automatically
void timerISR() {

    // This is an interrupt service routine. Don't put any heavy
     calculations here
    // or call anything that's not interrupt-safe, such as:
    // Serial, String, any memory allocation (new, malloc, etc.),
     Particle.publish and other Particle methods, and more.
    if(writer == 0){
        SampleBuf *sb = &buffers[sampleIndex % 2];

        if (!sb->free) {
            // The network has fallen behind in sending so discard samples
            return;
        }

      sample = analogRead(CONFIG.MICRO_PIN)-ADC_CENTER;
        // Convert 12-bit sample to 16-bit
      sb->data[sb->index++] = (uint8_t)(sample & 0xFF);
        sb->data[sb->index++] = (uint8_t)((sample>>8) & 0xFF);

        if (sb->index >= CONFIG.sampleBufferSize) {
            // Buffer has been filled.
            sb->free = false;
            sb->index = 0;
            sampleIndex++;
      }
    }
}
```

A.2 Node Server Project

```
const http = require('http');
const app = require('./app');

const port = process.env.PORT || 8080;

const server = http.createServer(app);

server.listen(port);const mongoose = require("mongoose");

const dataSchema = mongoose.Schema({
  _id: String,
  deviceId: String,
  date: String,

  latitude: Number,
  longitude: Number,

  temperature: Number,
  depth: Number,
  altitude: Number,
  pressure: Number,
  turbidity: Number,
  ph: Number,
  oxygen: Number
});

module.exports = mongoose.model("data", dataSchema);
const express = require("express");
const router = express.Router();
const mongoose = require("mongoose");

const DataModel = require("../models/dataModel");

//handler for the GET request
router.get("/", (req, res, next) => {
  DataModel.find(
    {},
    {
      _id: 1,
      deviceId: 1,
      date: 1,

      latitude: 1,
      longitude: 1,

      temperature: 1,
      depth: 1,
      altitude: 1,
      pressure: 1,
      turbidity: 1,
      ph: 1,
      oxygen: 1
    }
  )
    .exec()
    .then(doc => {
      console.log(doc);
      res.status(200).json(doc);
    })
    .catch(error => {
      console.log(error);
    });
});

// handler for the POST request
router.post("/", (req, res, next) => {
  let array = req.body;

  console.log(array);

  array.forEach(element => {
    //checks to see if the id already exists
    DataModel.findById(element.id)
      .exec()
      .then(doc => {
        //if not saves the data
        if (doc == undefined) {
          const data = new DataModel({
            _id: element.id,
            deviceId: element.deviceId,
            date: element.date,

            latitude: element.latitude,
            longitude: element.longitude,

            temperature: element.temperature,
            depth: element.depth,
            altitude: element.altitude,
            pressure: element.pressure,
            turbidity: element.turbidity,
            ph: element.ph,
            oxygen: element.oxygen
          });

          //console.log(data);

          data.save().catch(error => {
            console.log(" error saving data");
          });
        } else {
          console.log("data already exists");
        }
      });
  });

  res.status(200).json({
    message: "Data Received"
  });
});
```

```
module.exports = router;
// Imports
const express = require('express');
const app = express();
const morgan = require('morgan');
const bodyParser = require('body-parser');
const mongoose = require('mongoose');

// Routes Import
const dataRoutes = require('./api/routes/data');

mongoose.connect('mongodb://127.0.0.1/whalesDb');

// for dev logs of requests
app.use(morgan('dev'));
app.use(bodyParser.urlencoded({
    extended: true
}));
app.use(bodyParser.json());

app.use((req, res, next) => {
    res.header("Access-Control-Allow-Origin", "*");
    res.header(
        "Access-Control-Allow-Headers",
        "Origin, X-Requested-With, Content-Type, Accept, Authorization"
    );
    if (req.method === 'OPTIONS') {
        res.header('Access-Control-Allow-Methods', 'PUT, POST, PATCH,
         DELETE, GET');
        return res.status(200).json({});
    }
    next();
});

// Routes that handle requests
app.use('/data', dataRoutes); // Everything that starts with /data comes
 here

// Error found throw it (Example: Delete methos not supported by the API)
app.use((req, res, next) => {
    const error = new Error('Not found');
    error.status = 404;
    next(error);
})

// Sends the Error to the client
app.use((error, req, res, next) => {
    res.status(error.status || 500);
    res.json({
        error: {
            message: error.message
        }
    });
})
```

```
module.exports = app;
```

A.3 iOS Application Project

```swift
//
//  DbConnection.swift
//  WhalesAppIOS
//
//  Created by Pedro Ribeiro on 04/05/2018.
//  Copyright © 2018 Pedro Ribeiro. All rights reserved.
//

import Foundation
import SwiftyJSON
import RealmSwift

class DbConnection {
    let realm = try! Realm()

    public func saveModelToDb(obj: DataModel){
        try! realm.write {
            realm.add(obj, update: true)
        }
    }

    public func getDataFromDb() -> [DataModel] {
        return realm.objects(DataModel.self).map({$0 as DataModel}).sorted()
         {$0.date < $1.date }
    }

    public func getDataFromDb(isFromServer: Bool) -> [DataModel] {
        return realm.objects(DataModel.self).filter("isFromServer = \
         (isFromServer)") .map({$0 as DataModel}).sorted(){$0.date < $1.date
         }
    }

    public func checkIfObjectExists(id: String) -> Bool {
        if realm.object(ofType: DataModel.self, forPrimaryKey: id) != nil {
            return true
        }
        return false
    }

    public func updateData(id: String, isFromServer: Bool) -> Bool{
        try! realm.write {
            realm.create(DataModel.self, value: ["id": id, "isFromServer":
             isFromServer], update: true)
        }
        return true
    }

    public func updateDepth(id: String, depth: Double) -> Bool{
        try! realm.write {
            realm.create(DataModel.self, value: ["id": id, "depth": depth],
             update: true)
        }
        return true
    }

    //-------------------- To Manage Data easly --------------//

    ///gets the data for one day
    public func getDateForDay(currentDate: Date) -> [DataModel] {
        //connects to the database to get the data
        let db = DbConnection()
        var array = db.getDataFromDb()

        array = array.filter { Calendar.current.isDate($0.date, equalTo:
         currentDate, toGranularity: .day)}

        // if there is no data just return nil
        if array.isEmpty {
            return [DataModel]()
        }

        let dict = Dictionary(grouping: array) {$0.date.hour}

        var finalArray = [DataModel]()

        let component =
         Calendar.current.dateComponents([.year, .month, .day], from:
         currentDate)
        var d = Calendar.current.date(from: component)!

        for _ in 0..<24 {
            let obj = DataModel()
            obj.date = d
            finalArray.append(obj)
            d = Calendar.current.date(byAdding: .hour, value: 1, to: d)!
        }

        for (_, value) in dict {
            for obj in finalArray {
                for v in value {
                    if(obj.date.hour == v.date.hour){
                        obj.add(obj: v)
                    }
                }
                if(value.count > 0 && value[0].date.hour == obj.date.hour ){
                    obj.divide(value: value.count)
                }
            }
        }
        return finalArray
    }

    ///gets the data for one week all days grouped in average
    public func getDateForWeek(currentDate: Date) -> [DataModel] {
        //connects to the database to get the data
        let db = DbConnection()
        var array = db.getDataFromDb()
```

```
    array = array.filter { Calendar.current.isDate($0.date, equalTo:
     currentDate, toGranularity: .weekOfYear )}


    // if there is no data just return nil
    if array.isEmpty {
        return [DataModel]()
    }

    let dict = Dictionary(grouping: array) {$0.date.day}

    var finalArray = [DataModel]()

    let component =
     Calendar.current.dateComponents([.year, .month, .day, .hour, .minut
     e, .second], from: currentDate.startOfWeek!)
    var d = Calendar.current.date(from: component)!

    for _ in 0..<7 {
        let obj = DataModel()
        obj.date = d
        finalArray.append(obj)
        d = Calendar.current.date(byAdding: .day, value: 1, to: d)!
    }

    for (_, value) in dict {
        for obj in finalArray {
            for v in value {
                if(obj.date.day == v.date.day){
                    obj.add(obj: v)
                }
            }
            if(value.count > 0 && value[0].date.day == obj.date.day ){
                obj.divide(value: value.count)
            }
        }
    }
    return finalArray
}

///gets the data for one month all days grouped in average
public func getDateForMonth(currentDate: Date) -> [DataModel] {
    //connects to the database to get the data
    let db = DbConnection()
    var array = db.getDataFromDb()

    array = array.filter { Calendar.current.isDate($0.date, equalTo:
     currentDate, toGranularity: .month)}

    // if there is no data just return nil
    if array.isEmpty {
        return [DataModel]()
    }

    let dict = Dictionary(grouping: array) {$0.date.day}

    var finalArray = [DataModel]()

    let range = Calendar.current.range(of: .day, in: .month, for:
     currentDate)!

    var component =
     Calendar.current.dateComponents([.year, .month, .day, .hour, .minut
     e, .second], from: currentDate)
    component.day = 1
    var d = Calendar.current.date(from: component)!

    for _ in 0..<range.count {
        let obj = DataModel()
        obj.date = d
        finalArray.append(obj)
        d = Calendar.current.date(byAdding: .day, value: 1, to: d)!
    }

    for (key, value) in dict {
        for v in value {
            finalArray[key-1].add(obj: v)
        }
        finalArray[key-1].divide(value: value.count)
    }
    return finalArray
}

///gets the data for one year all grouped in average
public func getDateForYear(currentDate: Date) -> [DataModel] {
    var array = self.getDataFromDb()

    array = array.filter { Calendar.current.isDate($0.date, equalTo:
     currentDate, toGranularity: .year)}

    // if there is no data just return nil
    if array.isEmpty {
        return [DataModel]()
    }

    let dict = Dictionary(grouping: array) {$0.date.month}

    //initialize the array with fixed size
    var finalArray = [DataModel]()
    var component =
     Calendar.current.dateComponents([.year, .month, .day, .hour, .minut
     e, .second], from: currentDate)
    component.month = 1
    component.day = 1
    var d = Calendar.current.date(from: component)!
    for _ in 0..<12 {
        let obj = DataModel()
        obj.date = d
        finalArray.append(obj)
        d = Calendar.current.date(byAdding: .month, value: 1, to: d)!
```

```
        }

        for (key, value) in dict {
            for v in value {
                finalArray[key-1].add(obj: v)
            }
            finalArray[key-1].divide(value: value.count)
        }

        return finalArray
    }
}


//
//  DataModel.swift
//  WhalesAppIOS
//
//  Created by Pedro Ribeiro on 30/05/2018.
//  Copyright © 2018 Pedro Ribeiro. All rights reserved.
//


import RealmSwift
import SwiftyJSON

///Object that is used to save the data to Realm and also use it across the
 application
class DataModel: Object {

    /// init of the object used when receiving from server and device
    convenience init(isFromServer: Bool, deviceId: String, date: String,
     latitude: Double, longitude: Double, temperature: Double, depth:
     Double, altitude: Double, pressure: Double, turbidity: Double, ph:
     Double, oxygen: Double) {
        self.init()

        self.id = deviceId + date
        self.isFromServer = isFromServer

        self.date = DateFormatter().stringToDate(str: date)!
        self.deviceId = deviceId

        self.latitude = latitude
        self.longitude = longitude

        self.temperature = temperature
        self.depth = abs(depth)
        self.altitude = altitude
        self.pressure = pressure
        self.turbidity = turbidity
        self.ph = ph
        self.oxygen = oxygen

        isNull = false
    }


    @objc dynamic var id = ""
    @objc dynamic var isFromServer = false

    @objc dynamic var date = Date()
    @objc dynamic var deviceId = ""


    @objc dynamic var latitude: Double = 0.0
    @objc dynamic var longitude: Double = 0.0


    @objc dynamic var temperature: Double = 0.0
    @objc dynamic var depth: Double = 0.0
    @objc dynamic var altitude: Double = 0.0
    @objc dynamic var pressure: Double = 0.0
    @objc dynamic var turbidity: Double = 0.0
    @objc dynamic var ph: Double = 0.0
    @objc dynamic var oxygen: Double = 0.0

    //variable used to know if the object is a placeholder (ex average of
     real data)
    var isNull = true

    ///returns the primary key id
    override static func primaryKey() -> String? {
        return "id"
    }

    //returns coordinates in Lat:..., Long:...
    var coordinates: String {
        get {
            return "Lat: \(self.latitude), Long: \(self.longitude)"
        }
    }

    ///returns an array of strings with all the information
    var array: [String] {
        get {
            var array = [String]()
            array.append(String(format: "Lat: %.04f", self.latitude))
            array.append(String(format: "Long: %.04f", self.longitude))
            array.append(String(format: "Temperature: %.02f °C",
             self.temperature))
            array.append(String(format: "Depth: %.02f m", self.depth))
            array.append(String(format: "Altitude: %.02f m", self.altitude))
            array.append(String(format: "Pressure: %.02f bar",
             self.pressure))
            array.append(String(format: "Turbidity: %d", self.turbidity))
            array.append(String(format: "Ph: %.02f", self.ph))
            array.append(String(format: "Oxygen: %.02f mg/L", self.oxygen))
            return array
        }
```

```
}

/// converts the object into JSON to send to the server
public func toJson()->JSON {
    var json = JSON()

    json["id"] = JSON(id)
    json["deviceId"] = JSON(deviceId)
    json["date"] = JSON(ISO8601DateFormatter().string(from: date))


    json["latitude"] = JSON(latitude)
    json["longitude"] = JSON(longitude)

    json["temperature"] = JSON(temperature)
    json["depth"] = JSON(depth)
    json["altitude"] = JSON(altitude)
    json["pressure"] = JSON(pressure)
    json["turbidity"] = JSON(turbidity)
    json["ph"] = JSON(ph)
    json["oxygen"] = JSON(oxygen)

    return json
}

/// used to group objects to then calculate the average
public func add(obj: DataModel){

    //standard
    self.deviceId = obj.deviceId
    self.id = obj.id
    self.latitude = obj.latitude
    self.longitude = obj.longitude
    //self.date = obj.date

    //add values
    self.temperature += obj.temperature
    self.depth += obj.depth
    self.altitude += obj.altitude
    self.pressure += obj.pressure
    self.turbidity += obj.turbidity
    self.ph += obj.ph
    self.oxygen += obj.oxygen

    self.isNull = false
}

/// used to divide the values in order to obtain the average
public func divide(value: Int){
    self.temperature /= Double(value)
    self.depth /= Double(value)
    self.altitude /= Double(value)
    self.pressure /= Double(value)
    self.turbidity /= Double(value)
    self.ph /= Double(value)
    self.oxygen /= Double(value)
    }
}
//
//  ConnectToDevice.swift
//  WhalesAppIOS
//
//  Created by Pedro Ribeiro on 08/05/2018.
//  Copyright © 2018 Pedro Ribeiro. All rights reserved.
//

import Foundation
import Alamofire
import SwiftyJSON

class ConnectionToDevice {

    let baseUrlString = "http://192.168.0.1/data?";
    var upcomingDataDictionary = [String: String]()    // deviceId; csvData


    /// gets the data from the photon
    func getData(i: Int){

        // url with id of the get part
        let url = URL(string: baseUrlString + String(i))

        Alamofire.request(url!)
            .validate()
            .responseJSON{ response in
                switch response.result {
                case .success(let value):
                    let json = JSON(value)

                    // save the data to the dictionairy
                    // in this case there is data already with that deviceID
                    if let val =
                     self.upcomingDataDictionary[json["deviceId"].string!] {

                         self.upcomingDataDictionary[json["deviceId"].string
                         !] = val + json["csvData"].string!
                    }
                        // in this case there is no data with the device id
                         asked
                    else {

                         self.upcomingDataDictionary[json["deviceId"].string
                         !] = json["csvData"].string!
                    }

                    // check if this is the last piece of data
                    if(json["isFinalData"].boolValue) {
                        print("here")
                        self.convertCSVDataAndSaveToDb(deviceId:
                         json["deviceId"].string!)
```

```
                    print("all the data from photon is saved")
                } else {
                    print(json["dataPart"].int!)
                    self.getData(i: json["dataPart"].int!)
                }
            case .failure(let error):
                print(error)
            }
    }

}

/// convert all the data received from the photon and saves it into the
 database
private func convertCSVDataAndSaveToDb(deviceId: String){
    let db = DbConnection()
    let csv = upcomingDataDictionary[deviceId]!
    var linesArray = csv.split(separator: ";")

    //getting the headers
    let headers = linesArray[0].split(separator: ",")

    //remove first element from the array -> headers
    linesArray.remove(at: 0)

    for line in linesArray {
        let currentLine = line.split(separator: ",")
        var jsonObj = JSON()
        jsonObj["deviceId"] = JSON(deviceId)

        for index in 0..<headers.count {
            let header = String(headers[index])

            switch  header{
            case "date":
                jsonObj[header] = JSON(String(currentLine[index]))
                break
            default:
                jsonObj[header] =
                 JSON(Double(String(currentLine[index]))!)
            }
        }

        if(!db.checkIfObjectExists(id:deviceId +
         jsonObj["date"].string!)){
```

```
                let dataObj = DataModel(isFromServer: false, deviceId:
                 deviceId, date: jsonObj["date"].string!, latitude:
                 jsonObj["latitude"].double!, longitude:
                 jsonObj["longitude"].double!, temperature:
                 jsonObj["temperature"].double!, depth:
                 jsonObj["depth"].double!, altitude:
                 jsonObj["altitude"].double!, pressure:
                 jsonObj["pressure"].double!, turbidity:
                 jsonObj["turbidity"].double!, ph: jsonObj["ph"].double!,
                 oxygen: jsonObj["oxygen"].double!)

                db.saveModelToDb(obj: dataObj)
            }

        }
    }
}
//
//  MapAnnotation.swift
//  WhalesAppIOS
//
//  Created by Pedro Ribeiro on 04/06/2018.
//  Copyright © 2018 Pedro Ribeiro. All rights reserved.
//

import Foundation
import MapKit

class MapAnnotation: NSObject, MKAnnotation {
    var coordinate: CLLocationCoordinate2D

    let title: String?
    let data: DataModel
    let pinCustomImageName = "whaleIcon"


    init(title: String, data: DataModel, coordinate: CLLocationCoordinate2D)
     {
        self.title = title
        self.data = data
        self.coordinate = coordinate

        super.init()
    }

}
//
//  ConnectionToServer.swift
//  WhalesAppIOS
//
//  Created by Pedro Ribeiro on 04/05/2018.
//  Copyright © 2018 Pedro Ribeiro. All rights reserved.
//

import Foundation
```

```
import Alamofire
import SwiftyJSON


class ConnectionToServer {
    private let serverUrl = URL(string: "http://192.168.0.112:8080/data")

    public func sendDataToServer(){
        //checks to see if there is internte connection
        if !NetworkReachabilityManager()!.isReachable {
            print("Network not available")
            return
        }

        let db = DbConnection();

        let arrayData = db.getDataFromDb(isFromServer: false)
        print(arrayData.count)
        if !arrayData.isEmpty {

            var request = URLRequest(url: serverUrl!)
            request.httpMethod = HTTPMethod.post.rawValue
            request.setValue("application/json", forHTTPHeaderField:
             "Content-Type")

            var array = [JSON]()
            for obj in arrayData {
                array.append(obj.toJson())

            }
            let data = (JSON(array).rawString()! .data(using: .utf8))! as
             Data

            print(JSON(array).rawString()!)
            request.httpBody = data

            Alamofire.request(request).responseJSON { (response) in
                switch response.result {
                case .success(let value):
                    print(value)

                    for obj in arrayData {
                        if !db.updateData(id: obj.id, isFromServer: true){
                            print("can't update data in db")
                        }
                    }
                case .failure(let error):
                    print(error)
                }
            }
        }
        else {
            print("There is no data to send to the server")
        }
    }

    public func getDataFromServer(completion : @escaping ()->()) {
        //checks to see if there is internte connection
        if !NetworkReachabilityManager()!.isReachable {
            print("Network not available")
            return
        }


        Alamofire.request(serverUrl!)
            .validate()
            .responseJSON{ response in
                switch response.result {
                case .success(let value):
                    let db = DbConnection();
                    let json = JSON(value)
                    for (_,jsonObj):(String, JSON) in json {

                        let dataObj = DataModel(isFromServer: true,
                         deviceId: jsonObj["deviceId"].string!, date:
                         jsonObj["date"].string!, latitude:
                         jsonObj["latitude"].double!, longitude:
                         jsonObj["longitude"].double!, temperature:
                         jsonObj["temperature"].double!, depth:
                         jsonObj["depth"].double!, altitude:
                         jsonObj["altitude"].double!, pressure:
                         jsonObj["pressure"].double!, turbidity:
                         jsonObj["turbidity"].double!, ph:
                         jsonObj["ph"].double!, oxygen:
                         jsonObj["oxygen"].double!)

                        db.saveModelToDb(obj: dataObj)
                    }

                    completion()
                case .failure(let error):
                    completion()
                    print(error)
                }
        }
    }
}


//
//  AppDelegate.swift
//  WhalesAppIOS
//
//  Created by Pedro Ribeiro on 04/05/2018.
//  Copyright © 2018 Pedro Ribeiro. All rights reserved.
//

import UIKit

@UIApplicationMain
```

```
class AppDelegate: UIResponder, UIApplicationDelegate {

    var window: UIWindow?



    func application(_ application: UIApplication,
     didFinishLaunchingWithOptions launchOptions:
     [UIApplicationLaunchOptionsKey: Any]?) -> Bool {
        // Override point for customization after application launch.

        return true
    }

    func applicationWillResignActive(_ application: UIApplication) {
        // Sent when the application is about to move from active to
         inactive state. This can occur for certain types of temporary
         interruptions (such as an incoming phone call or SMS message) or
         when the user quits the application and it begins the transition to
         the background state.
        // Use this method to pause ongoing tasks, disable timers, and
         invalidate graphics rendering callbacks. Games should use this
         method to pause the game.
    }

    func applicationDidEnterBackground(_ application: UIApplication) {
        // Use this method to release shared resources, save user data,
         invalidate timers, and store enough application state information
         to restore your application to its current state in case it is
         terminated later.
        // If your application supports background execution, this method is
         called instead of applicationWillTerminate: when the user quits.
    }

    func applicationWillEnterForeground(_ application: UIApplication) {
        // Called as part of the transition from the background to the
         active state; here you can undo many of the changes made on
         entering the background.
    }

    func applicationDidBecomeActive(_ application: UIApplication) {
        // Restart any tasks that were paused (or not yet started) while the
         application was inactive. If the application was previously in the
         background, optionally refresh the user interface.
    }

    func applicationWillTerminate(_ application: UIApplication) {
        // Called when the application is about to terminate. Save data if
         appropriate. See also applicationDidEnterBackground:.
    }


}

//
```

```
//  MonthValueFormatter.swift
//  WhalesAppIOS
//
//  Created by Pedro Ribeiro on 19/06/2018.
//  Copyright © 2018 Pedro Ribeiro. All rights reserved.
//

import Foundation
import Charts

public class YearValueFormatter: NSObject, IAxisValueFormatter {
    public func stringForValue(_ value: Double, axis: AxisBase?) -> String {
        // to prevent error
        var x = Int(value)
        if(x > 11){
            x = 11
        }
        return Calendar.current.shortMonthSymbols[x]
    }
}
//
//  MonthValueFormatter.swift
//  WhalesAppIOS
//
//  Created by Pedro Ribeiro on 19/06/2018.
//  Copyright © 2018 Pedro Ribeiro. All rights reserved.
//

import Foundation
import Charts

public class WeekValueFormatter: NSObject, IAxisValueFormatter {

    private var initialWeekDate: Date = Date()

    init(initialWeekDate: Date) {
        super.init()
        self.initialWeekDate = initialWeekDate
    }

    public func stringForValue(_ value: Double, axis: AxisBase?) -> String {
        let d = Calendar.current.date(byAdding: .day, value: Int(value), to:
         initialWeekDate)!
        return "\(d.day)/\(d.month)"
    }
}
//
//  MonthValueFormatter.swift
//  WhalesAppIOS
//
//  Created by Pedro Ribeiro on 19/06/2018.
//  Copyright © 2018 Pedro Ribeiro. All rights reserved.
//

import Foundation
```

```swift
import Charts

public class MonthValueFormatter: NSObject, IAxisValueFormatter {

    public func stringForValue(_ value: Double, axis: AxisBase?) -> String {
        return String(format: "%.0f", value+1)
    }
}
//
//  DateValueFormatter.swift
//  WhalesAppIOS
//
//  Created by Pedro Ribeiro on 05/05/2018.
//  Copyright © 2018 Pedro Ribeiro. All rights reserved.
//

import Foundation
import Charts

public class DateValueFormatter: NSObject, IAxisValueFormatter {
    private let dateFormatter = DateFormatter()

    init(format: String) {
        super.init()
        dateFormatter.dateFormat = format
    }

    public func stringForValue(_ value: Double, axis: AxisBase?) -> String {
        return dateFormatter.string(from: Date(timeIntervalSince1970:
         value))
    }
}
//
//  MonthValueFormatter.swift
//  WhalesAppIOS
//
//  Created by Pedro Ribeiro on 19/06/2018.
//  Copyright © 2018 Pedro Ribeiro. All rights reserved.
//

import Foundation
import Charts

public class DayValueFormatter: NSObject, IAxisValueFormatter {

    public func stringForValue(_ value: Double, axis: AxisBase?) -> String {
        return String(format: "%.0f", value + 1)
    }
}
//
//  Extensions.swift
//  WhalesAppIOS
//
//  Created by Pedro Ribeiro on 04/05/2018.
//  Copyright © 2018 Pedro Ribeiro. All rights reserved.
//

import Foundation
import SwiftyJSON
import RealmSwift

extension DateFormatter {

    func jsonDateToDate(str: String) -> Date? {
        let formatter = DateFormatter()
        formatter.dateFormat = "yyyy-MM-dd'T'HH:mm:ss.SSSZ"
        let yourDate = formatter.date(from: str)
        return yourDate
    }

    func stringToDate(str: String) -> Date? {
        let formatter = DateFormatter()
        formatter.dateFormat = "yyyy-MM-dd'T'HH:mm:ssZ"
        let yourDate = formatter.date(from: str)
        return yourDate
    }
}

extension Date {

    var hour: Int {
        return Calendar.current.component(.hour, from: self)
    }

    var weekDay : Int {
        return Calendar.current.component(.weekday, from: self)
    }

    var day: Int {
        return Calendar.current.component(.day, from: self)
    }

    var month: Int {
        return Calendar.current.component(.month, from: self)
    }

    var year: Int {
        return Calendar.current.component(.year, from: self)
    }

    var startOfWeek: Date? {
        let gregorian = Calendar(identifier: .gregorian)
        guard let sunday = gregorian.date(from:
         gregorian.dateComponents([.yearForWeekOfYear, .weekOfYear], from:
         self)) else { return nil }
        return sunday
    }

    var endOfWeek: Date? {
        let gregorian = Calendar(identifier: .gregorian)
```

```
        guard let sunday = gregorian.date(from:
         gregorian.dateComponents([.yearForWeekOfYear, .weekOfYear], from:
         self)) else { return nil }
        return gregorian.date(byAdding: .day, value: 7, to: sunday)
    }

    func toString(withFormat format: String) -> String {
        let formatter = DateFormatter()
        formatter.dateFormat = format
        let myString = formatter.string(from: self)
        let yourDate = formatter.date(from: myString)
        formatter.dateFormat = format

        return formatter.string(from: yourDate!)
    }

    func startOfMonth() -> Date {
        return Calendar.current.date(from:
         Calendar.current.dateComponents([.year, .month], from:
         Calendar.current.startOfDay(for: self)))!
    }

    func endOfMonth() -> Date {
        return Calendar.current.date(byAdding: DateComponents(month: 1, day:
         -1), to: self.startOfMonth())!
    }


}

extension UIColor {
    static var mainPink = UIColor(red: 232/255, green: 68/255, blue:
     133/255, alpha: 1)
}

extension UIView {
    func addConstraintsWithFormat(format: String, views: UIView...){
        var viewsDictionairy = [String: UIView]()
        for (index, view) in views.enumerated(){
            let key = "v\(index)"
            view.translatesAutoresizingMaskIntoConstraints = false
            viewsDictionairy[key] = view
        }

        addConstraints(NSLayoutConstraint.constraints(withVisualFormat:
         format, options: NSLayoutFormatOptions(), metrics: nil, views:
         viewsDictionairy))
    }
}

extension UIColor {
    func toHexString() -> String {
        var r:CGFloat = 0
        var g:CGFloat = 0
        var b:CGFloat = 0
        var a:CGFloat = 0

        getRed(&r, green: &g, blue: &b, alpha: &a)

        let rgb:Int = (Int)(r*255)<<16 | (Int)(g*255)<<8 | (Int)(b*255)<<0

        return String(format:"#%06x", rgb)
    }
}

//
//  Formatter.swift
//  WhalesAppIOS
//
//  Created by Pedro Ribeiro on 04/05/2018.
//  Copyright © 2018 Pedro Ribeiro. All rights reserved.
//

import Foundation

class Formatter {

    private static var internalJsonDateFormatter: DateFormatter?
    private static var internalJsonDateTimeFormatter: DateFormatter?

    static var jsonDateFormatter: DateFormatter {
        if (internalJsonDateFormatter == nil) {
            internalJsonDateFormatter = DateFormatter()
            internalJsonDateFormatter!.dateFormat = "yyyy-MM-dd"
        }
        return internalJsonDateFormatter!
    }

    static var jsonDateTimeFormatter: DateFormatter {
        if (internalJsonDateTimeFormatter == nil) {
            internalJsonDateTimeFormatter = DateFormatter()
            internalJsonDateTimeFormatter!.dateFormat =
             "yyyy'-'MM'-'dd'T'HH':'mm':'ss.SSS'Z'"
        }
        return internalJsonDateTimeFormatter!
    }

}
//
//  Errors.swift
//  WhalesAppIOS
//
//  Created by Pedro Ribeiro on 05/05/2018.
//  Copyright © 2018 Pedro Ribeiro. All rights reserved.
//

import Foundation

enum MyErrors: Error {
    case DiferentNumberOfElements(msg: String)
```

```
}
//
//  PopupLauncher.swift
//  WhalesAppIOS
//
//  Created by Pedro Ribeiro on 06/08/2018.
//  Copyright © 2018 Pedro Ribeiro. All rights reserved.
//

import UIKit

class PopupLauncher: NSObject, UICollectionViewDataSource,
 UICollectionViewDelegate, UICollectionViewDelegateFlowLayout {

    let blackView = UIView()

    let collectionView: UICollectionView = {
        let layout = UICollectionViewFlowLayout()
        let cv = UICollectionView(frame: .zero, collectionViewLayout:
         layout)

        cv.backgroundColor = UIColor(red: 239, green: 239, blue: 244, alpha:
         1)
        return cv
    }()

    let cellId = "cellId"

    var arrayCells: [CellInfo] = {
        return [
            CellInfo(iconName: "date", text: ""),
            CellInfo(iconName: "gps", text: ""),
            CellInfo(iconName: "temperature", text: ""),
            CellInfo(iconName: "depth", text: ""),
            CellInfo(iconName: "pressure", text: ""),
            CellInfo(iconName: "turbidity", text: ""),
            CellInfo(iconName: "pH", text: ""),
            CellInfo(iconName: "oxygen", text: ""),
        ]
    }()

    override init(){
        super.init()

        collectionView.dataSource = self
        collectionView.delegate = self

        collectionView.register(PopupCell.self, forCellWithReuseIdentifier:
         cellId)
    }


    /// launches all the popup and background
    func handleBottomPopup(model: DataModel){
        if let window = UIApplication.shared.keyWindow {
            blackView.backgroundColor = UIColor(white: 0, alpha: 0.5)

            blackView.addGestureRecognizer(UITapGestureRecognizer(target:
             self, action: #selector(handleDismiss)))

            window.addSubview(blackView)
            window.addSubview(collectionView)

            blackView.frame = window.frame
            blackView.alpha = 0

            let height: CGFloat = 200
            let y = window.frame.height - height

            collectionView.frame = CGRect(x: 0, y: window.frame.height,
             width: window.frame.width, height: height)


            UIView.animate(withDuration: 0.5, delay: 0,
             usingSpringWithDamping: 1, initialSpringVelocity: 1,
             options: .curveEaseOut, animations: {
                    self.blackView.alpha = 1
                    self.collectionView.frame = CGRect(x: 0, y: y, width:
                     window.frame.width, height: height)
                    }, completion: nil)

            arrayCells[0].text = model.date.toString(withFormat: "yyyy-MM-dd
             HH:mm")
            arrayCells[1].text = String(format: "Lat: %.04f, Long: %.04f",
             model.latitude, model.longitude)
            arrayCells[2].text = String(format: "Temperature: %.02f °C",
             model.temperature)
            arrayCells[3].text = String(format: "Depth: %.02f m",
             model.depth)
            arrayCells[4].text = String(format: "Pressure: %.02f bar",
             model.pressure)
            arrayCells[5].text = String(format: "Turbidity: %d",
             model.turbidity)
            arrayCells[6].text = String(format: "Ph: %.02f", model.ph)
            arrayCells[7].text = String(format: "Oxygen: %.02f mg/L",
             model.oxygen)

            self.collectionView.reloadData()
        }

    }

    ///handles the dismiss of the popup
    @objc func handleDismiss(){
        UIView.animate(withDuration: 0.5, animations: {
            self.blackView.alpha = 0

            if let window = UIApplication.shared.keyWindow {
```

```
                self.collectionView.frame = CGRect(x: 0, y:
                 window.frame.height, width:
                 self.collectionView.frame.width, height:
                 self.collectionView.frame.height)
            }
        })
    }


    //Collection View methods
    func collectionView(_ collectionView: UICollectionView,
     numberOfItemsInSection section: Int) -> Int {
        return arrayCells.count
    }

    func collectionView(_ collectionView: UICollectionView, cellForItemAt
     indexPath: IndexPath) -> UICollectionViewCell {
        let cell = collectionView.dequeueReusableCell(withReuseIdentifier:
         cellId, for: indexPath) as! PopupCell

        cell.cellInfo = arrayCells[indexPath.item]

        return cell
    }

    func collectionView(_ collectionView: UICollectionView, layout
     collectionViewLayout: UICollectionViewLayout, sizeForItemAt indexPath:
     IndexPath) -> CGSize {
        return CGSize(width: collectionView.frame.width , height: 50)
    }
}

/// model object for popupcells
class CellInfo {
    let iconName: String
    var text: String

    init(iconName: String, text: String) {
        self.iconName = iconName
        self.text = text
    }
}
//
//  GpsViewController.swift
//  WhalesAppIOS
//
//  Created by Pedro Ribeiro on 29/05/2018.
//  Copyright © 2018 Pedro Ribeiro. All rights reserved.
//

import UIKit
import MapKit

class GpsViewController: UIViewController, MKMapViewDelegate {

    @IBOutlet weak var mapView: MKMapView!

    var selectedData: DataModel?

    weak var annotation: MapAnnotation?

    //radius of the region to view on start
    let regionRadius: CLLocationDistance = 10000

    // popupView
    let popupLauncher = PopupLauncher()

    override func viewDidLoad() {
        super.viewDidLoad()

        mapView.delegate = self

        let db = DbConnection()
        let arrayData = db.getDataFromDb()

        let initialLocation = CLLocationCoordinate2D(latitude:
         arrayData.last!.latitude, longitude: arrayData.last!.longitude)
        centerMapOnLocation(location: initialLocation)

        for obj in arrayData {
            let title = "Date: " + obj.date.toString(withFormat: "yyyy-MM-dd
             HH:mm:ss")
            let coordinate = CLLocationCoordinate2D(latitude: obj.latitude,
             longitude: obj.longitude)
            let annotation = MapAnnotation(title: title, data: obj,
             coordinate: coordinate)
            mapView.addAnnotation(annotation)
        }

    }

    /// centers the map on a given location
    func centerMapOnLocation(location: CLLocationCoordinate2D) {
        let coordinateRegion = MKCoordinateRegionMakeWithDistance(location,

                                                  regionRadius,
                                                  regionRadius)
        mapView.setRegion(coordinateRegion, animated: true)
    }

    /// generates a pin for each location
    func mapView(_ mapView: MKMapView, viewFor annotation: MKAnnotation) ->
     MKAnnotationView? {
        if annotation is MKUserLocation { return nil }
        let reuseIdentifier = "pin"
        var annotationView =
         mapView.dequeueReusableAnnotationView(withIdentifier:
         reuseIdentifier)
```

```
        if annotationView == nil {
            annotationView = MKPinAnnotationView(annotation: annotation,
             reuseIdentifier: reuseIdentifier)
            annotationView?.canShowCallout = false

        } else {
            annotationView!.annotation = annotation
        }
        return annotationView
    }

    /// creates a popupLauncher when a location is touched
    func mapView(_ mapView: MKMapView, didSelect view: MKAnnotationView) {
        if view is MKPinAnnotationView {
            if let annotation = view.annotation as? MapAnnotation {
                popupLauncher.handleBottomPopup(model: annotation.data)
            }
        }
    }




}



//
//  CubijsmViewController.swift
//  WhalesAppIOS
//
//  Created by Pedro Ribeiro on 29/05/2018.
//  Copyright © 2018 Pedro Ribeiro. All rights reserved.
//

import UIKit
import Charts

class CubijsmViewController: UIViewController {

    @IBOutlet weak var cubiChartView: LineChartView!


    @IBOutlet weak var segmentedControl: UISegmentedControl!
    @IBOutlet weak var backButton: UIButton!
    @IBOutlet weak var forwardButton: UIButton!
    @IBOutlet weak var dateLabel: UILabel!
    @IBOutlet var switches: [UISwitch]!


    private var currentDate = Date()
    private var arrayOfData = [DataModel]()


    private var arrayOfColors = [[UIColor(red:0.91, green:0.28, blue:0.33,
     alpha:1.0)], [UIColor(red:0.02, green:0.59, blue:1.00, alpha:1.0)],
     [UIColor(red:0.95, green:0.91, blue:0.31, alpha:1.0)]]


    override func viewDidLoad(){
        super.viewDidLoad()

        cubiChartView.delegate = self

        updateLabel()
        setChartOptions()
        drawCubicChartYear()

    }

    // Actions

    @IBAction func segmentedControllChanged(_ sender: UISegmentedControl) {
        switch sender.selectedSegmentIndex {
        case 0:
            drawCubicChartDay()
            break
        case 1:
            drawCubicChartWeek()
        case 2:
            drawCubicChartMonth()
        case 3:
            drawCubicChartYear()
        default:
            break
        }
        updateLabel()
    }

    @IBAction func backButtonPressed(_ sender: UIButton) {
        let calendar = Calendar.current
        switch segmentedControl.selectedSegmentIndex {
        case 0: //subtract a day in the currentDate
            currentDate = calendar.date(byAdding: .day, value: -1, to:
             currentDate)!
            drawCubicChartDay()
            break
        case 1: //subtract a week in the currentDate
            currentDate = calendar.date(byAdding: .day, value: -7, to:
             currentDate)!
            drawCubicChartWeek()
            break
        case 2: //subtract a month in the currentDate
            currentDate = calendar.date(byAdding: .month, value: -1, to:
             currentDate)!
            drawCubicChartMonth()
            break
        case 3: //subtract a year in the currentDate
```

```
        currentDate = calendar.date(byAdding: .year, value: -1, to:
         currentDate)!
        drawCubicChartYear()
        break
    default:
        break
    }
    updateLabel()
}


@IBAction func forwardButtonPressed(_ sender: UIButton) {
    let calendar = Calendar.current
    switch segmentedControl.selectedSegmentIndex {
    case 0: //add a day in the currentDate
        currentDate = calendar.date(byAdding: .day, value: 1, to:
         currentDate)!
        drawCubicChartDay()
        break
    case 1: //add a week in the currentDate
        currentDate = calendar.date(byAdding: .day, value: 7, to:
         currentDate)!
        drawCubicChartWeek()
        break
    case 2: //add a month in the currentDate
        currentDate = calendar.date(byAdding: .month, value: 1, to:
         currentDate)!
        drawCubicChartMonth()
        break
    case 3: //add a year in the currentDate
        currentDate = calendar.date(byAdding: .year, value: 1, to:
         currentDate)!
        drawCubicChartYear()
        break
    default:
        break
    }
    updateLabel()
}

///handels the change of the switches -> calls the method to update the
 data
@IBAction func switchChanged(_ sender: UISwitch) {
    var count = 0
    var lastSwitch: UISwitch?
    for swit in switches {
        if swit.isOn && swit != sender{
            count += 1
            lastSwitch = swit
        }
    }
    if count == 0 {
        sender.setOn(true, animated: false)
    }
    if count >= 3 {
        lastSwitch!.setOn(false, animated: true)
    }
    segmentedControllChanged(segmentedControl);
}


/// Updates the label of the current day
private func updateLabel(){
    let calendar = Calendar.current
    switch segmentedControl.selectedSegmentIndex {
        case 0: // Day
            if calendar.isDateInToday(currentDate){
                dateLabel.text = "Today"
            } else if calendar.isDateInYesterday(currentDate){
                dateLabel.text = "Yesterday"
            } else {
                dateLabel.text = currentDate.toString(withFormat: "dd/
                 MM/yyyy")
            }
            break
        case 1: // Week
            let endOfWeek = currentDate.endOfWeek!
            dateLabel.text =
             currentDate.startOfWeek!.toString(withFormat: "dd/MM/") + "
             - " + endOfWeek.toString(withFormat: "dd/MM/yyyy")
            break
        case 2: // Month
            dateLabel.text = currentDate.toString(withFormat: "MM/yyyy")
        case 3: // Year
            dateLabel.text = currentDate.toString(withFormat: "yyyy")
            break
        default:
            break
        }
}


// Chart Methods

private func drawCubicChartDay(){
    cubiChartView.data = nil

    arrayOfData = DbConnection().getDateForDay(currentDate: currentDate)

    //prevent from drawing if there is no data
    guard arrayOfData.count > 0 else {
        cubiChartView.noDataText = "There is no data for the current
         selected day"
        return
    }

    let data = selectDataToDisplay(arrayOfData: arrayOfData)

    //place this before data otherwise might get errors
    cubiChartView.xAxis.valueFormatter = DayValueFormatter()
```

```
    cubiChartView.data = data

    //always place after adding the data
    cubiChartView.xAxis.axisMaximum = 23
    cubiChartView.xAxis.axisMinimum = 0
    cubiChartView.setVisibleXRange(minXRange: 0, maxXRange: 23)
}

private func drawCubicChartWeek(){
    cubiChartView.data = nil

    arrayOfData = DbConnection().getDateForWeek(currentDate:
     currentDate)

    //prevent from drawing if there is no data
    guard arrayOfData.count > 0 else {
        cubiChartView.noDataText = "There is no data for the current
         selected week"
        return
    }

    let data = selectDataToDisplay(arrayOfData: arrayOfData)

    //place this before data otherwise might get errors
    cubiChartView.xAxis.valueFormatter =
     WeekValueFormatter(initialWeekDate: currentDate.startOfWeek!)

    cubiChartView.data = data

    //always place after adding the data
    cubiChartView.xAxis.axisMaximum = 6
    cubiChartView.xAxis.axisMinimum = 0
    cubiChartView.setVisibleXRange(minXRange: 6, maxXRange: 7)

}

private func drawCubicChartMonth(){
    cubiChartView.data = nil

     arrayOfData = DbConnection().getDateForMonth(currentDate:
      currentDate)

    //prevent from drawing if there is no data
    guard arrayOfData.count > 0 else {
        cubiChartView.noDataText = "There is no data for the current
         selected month"
        return
    }

    let data = selectDataToDisplay(arrayOfData: arrayOfData)

    //place this before data otherwise might get errors
    cubiChartView.xAxis.valueFormatter = MonthValueFormatter()
    cubiChartView.data = data

    //always place after adding the data
    cubiChartView.xAxis.axisMinimum = 0
    cubiChartView.xAxis.axisMaximum =
     Double(currentDate.endOfMonth().day - 1)
    cubiChartView.setVisibleXRange(minXRange: 0, maxXRange:
     Double(currentDate.endOfMonth().day) - 1)

}


private func drawCubicChartYear() {
    cubiChartView.data = nil

    arrayOfData = DbConnection().getDateForYear(currentDate:
     currentDate)

    // prevent from drawing if there is no data
    guard arrayOfData.count > 0 else {
        cubiChartView.noDataText = "There is no data for the current
         selected year"
        return
    }

    let data = selectDataToDisplay(arrayOfData: arrayOfData)

    //place this before data otherwise might get errors
    cubiChartView.xAxis.valueFormatter = YearValueFormatter()

    cubiChartView.data = data

    //always place after adding the data
    cubiChartView.xAxis.axisMaximum = 11
    cubiChartView.xAxis.axisMinimum = 0


}

private func selectDataToDisplay(arrayOfData: [DataModel]) ->
 LineChartData{
    var count = 0
    let data = LineChartData()
    for swit in switches {
        if(swit.accessibilityIdentifier! == "temperature"){
            if(swit.isOn){
                var lineChartEntry = [ChartDataEntry]()
                for obj in arrayOfData {
                    let value1 = ChartDataEntry(x:
                     convertDateToXBasedOnType(date: obj.date), y:
                     obj.temperature)
                    lineChartEntry.append(value1)
                }
```

```
            let line = lineChartGenerator(lineChartEntry:
             lineChartEntry, label: "temperature", color:
             arrayOfColors[count])
            count += 1
            data.addDataSet(line)
        }
    }
    if(swit.accessibilityIdentifier! == "depth"){
        if(swit.isOn){
            var lineChartEntry = [ChartDataEntry]()
            for obj in arrayOfData {
                let value1 = ChartDataEntry(x:
                 convertDateToXBasedOnType(date: obj.date), y:
                 obj.depth)
                lineChartEntry.append(value1)
            }
            let line = lineChartGenerator(lineChartEntry:
             lineChartEntry, label: "depth", color:
             arrayOfColors[count])
            count += 1
            data.addDataSet(line)
        }
    }
    if(swit.accessibilityIdentifier! == "pressure"){
        if(swit.isOn){
            var lineChartEntry = [ChartDataEntry]()
            for obj in arrayOfData {
                let value1 = ChartDataEntry(x:
                 convertDateToXBasedOnType(date: obj.date), y:
                 obj.pressure)
                lineChartEntry.append(value1)
            }
            let line = lineChartGenerator(lineChartEntry:
             lineChartEntry, label: "pressure", color:
             arrayOfColors[count])
            count += 1
            data.addDataSet(line)
        }
    }
    if(swit.accessibilityIdentifier! == "turbidity"){
        if(swit.isOn){
            var lineChartEntry = [ChartDataEntry]()
            for obj in arrayOfData {
                let value1 = ChartDataEntry(x:
                 convertDateToXBasedOnType(date: obj.date), y:
                 obj.turbidity)
                lineChartEntry.append(value1)
            }
            let line = lineChartGenerator(lineChartEntry:
             lineChartEntry, label: "turbidity", color:
             arrayOfColors[count])
            count += 1
            data.addDataSet(line)
        }
    }
    if(swit.accessibilityIdentifier! == "ph"){
        if(swit.isOn){
            var lineChartEntry = [ChartDataEntry]()
            for obj in arrayOfData {
                let value1 = ChartDataEntry(x:
                 convertDateToXBasedOnType(date: obj.date), y:
                 obj.ph)
                lineChartEntry.append(value1)
            }
            let line = lineChartGenerator(lineChartEntry:
             lineChartEntry, label: "ph", color:
             arrayOfColors[count])
            count += 1
            data.addDataSet(line)
        }
    }
    if(swit.accessibilityIdentifier! == "oxygen"){
        if(swit.isOn){
            var lineChartEntry = [ChartDataEntry]()
            for obj in arrayOfData {
                let value1 = ChartDataEntry(x:
                 convertDateToXBasedOnType(date: obj.date), y:
                 obj.oxygen)
                lineChartEntry.append(value1)
            }
            let line = lineChartGenerator(lineChartEntry:
             lineChartEntry, label: "oxygen", color:
             arrayOfColors[count])
            count += 1
            data.addDataSet(line)
        }
    }

    return data;
}

private func convertDateToXBasedOnType(date: Date) -> Double{
    switch segmentedControl.selectedSegmentIndex {
    case 0: // Day
        return Double(date.hour)
    case 1: // Week
        return Double(date.weekDay - 1)
    case 2: // Month
        return Double(date.day - 1)
    case 3: // Year
        return Double(date.month - 1)
    default:
        return 0
    }
}

///generates the line to then add to the graph
private func lineChartGenerator(lineChartEntry: [ChartDataEntry], label:
 String, color: [UIColor]) -> LineChartDataSet{
```

```
        let line = LineChartDataSet(values: lineChartEntry, label: label)
        line.colors = color

        line.mode = .horizontalBezier
        line.drawCirclesEnabled = false
        line.lineWidth = 1.8
        line.circleRadius = 4
        line.setCircleColor(.white)
        line.highlightColor = UIColor(red: 244/255, green: 117/255, blue:
         117/255, alpha: 1)

        line.drawFilledEnabled = true
        line.fillColor = color[0]

        line.drawHorizontalHighlightIndicatorEnabled = false

        // sets an area filled bellow the graph changed to zero to avoid
         that
        line.fillFormatter = CubicLineSampleFillFormatter()

        return line
    }


    ///sets some standard options to the graph just called once at the start
    private func setChartOptions(){

        cubiChartView.chartDescription?.enabled = false

        //disables x grid
        cubiChartView.xAxis.drawGridLinesEnabled = false

        //blocks zoom on Y axis
        cubiChartView.scaleYEnabled = false

        //animation
        cubiChartView.animate(xAxisDuration: 1)

        let leftAxis = cubiChartView.leftAxis
        leftAxis.xOffset = 10

        let rightAxis = cubiChartView.rightAxis
        rightAxis.enabled = false

        // xAxis defenitions
        let xAxis = cubiChartView.xAxis
        xAxis.labelPosition = .bottom
        xAxis.granularity = 1
    }

}

private class CubicLineSampleFillFormatter: IFillFormatter {

    func getFillLinePosition(dataSet: ILineChartDataSet, dataProvider:
     LineChartDataProvider) -> CGFloat {
        return 0
    }
}


extension CubijsmViewController: ChartViewDelegate {

    /// shows the information for the selected item on the chart
    public func chartValueSelected(_ chartView: ChartViewBase, entry:
     ChartDataEntry, highlight: Highlight) {
    }

}
//
//  CrimeaRoseViewController.swift
//  WhalesAppIOS
//
//  Created by Pedro Ribeiro on 29/05/2018.
//  Copyright © 2018 Pedro Ribeiro. All rights reserved.
//


import UIKit
import DatePickerDialog

class CrimeaRoseViewController: UIViewController {

    @IBOutlet weak var crimeaRoseView: CrimeaRoseView! {
        didSet {
            // sets the gestures recognizers for the view
            let pinch = UIPinchGestureRecognizer(target: crimeaRoseView,
             action: #selector(crimeaRoseView.didPinch(pinchGR:)))
            let panGR = UIPanGestureRecognizer(target: crimeaRoseView,
             action: #selector(crimeaRoseView.didPan(panGR:)))
            let rotationGR = UIRotationGestureRecognizer(target:
             crimeaRoseView, action:
             #selector(crimeaRoseView.didRotate(rotationGR:)))
            let tapGR = UITapGestureRecognizer(target: self, action:
             #selector(didTap(tapGR:)))

            crimeaRoseView.addGestureRecognizer(pinch)
            crimeaRoseView.addGestureRecognizer(panGR)
            crimeaRoseView.addGestureRecognizer(rotationGR)
            crimeaRoseView.addGestureRecognizer(tapGR)

        }
    }


    @IBOutlet weak var segmentedControl: UISegmentedControl!

    @IBOutlet weak var dateTypePicker: UIPickerView!
```

```
@IBOutlet weak var leftButtonDatePicker: UIButton!
@IBOutlet weak var rightButtonDatePicker: UIButton!

@IBOutlet weak var leftDateLabel: UILabel!
@IBOutlet weak var rightDateLabel: UILabel!


@IBOutlet weak var blueLabel: UILabel!
@IBOutlet weak var yellowLabel: UILabel!


private var currentDate = Date()

private var firstDateToCompare = Date()
private var secondDateToCompare = Date()

private let pickerData = ["Day", "Week", "Month", "Year"]


private let firstUIColor = UIColor(red:0.02, green:0.59, blue:1.00,
 alpha:1.0)
private let secondUIColor = UIColor(red:0.95, green:0.91, blue:0.31,
 alpha:1.0)

// arrays with the data
private var arrayLabels = [String]()
private var arrayOfCrimeaRoseData = [CrimeaRoseData]()
private var indexTouched = -1

override func viewDidLoad() {
    super.viewDidLoad()

    self.dateTypePicker.delegate = self
    self.dateTypePicker.dataSource = self

    leftDateLabel.backgroundColor = firstUIColor
    rightDateLabel.backgroundColor = secondUIColor

    dateTypePicker.selectRow(3, inComponent: 0, animated: false)

    firstDateToCompare = Date() // current Date
    secondDateToCompare = Calendar.current.date(byAdding: .year, value:
     0, to: firstDateToCompare)! // subtract one year to the current
     date

    updateLabels()
    drawChartForYear()

}

///draws the data to compare by day
private func drawChartForDay(){
    let arrayFirstdDay = DbConnection().getDateForDay(currentDate:
     firstDateToCompare)
    let arraySecondDay = DbConnection().getDateForDay(currentDate:
     secondDateToCompare)

    var arrayOfDataFirst: [Double]
    var arrayOfDataSecond: [Double]

    var (minValue, maxValue) = getMinMaxValue(minValue:
     Double.greatestFiniteMagnitude, maxValue:
     Double.leastNormalMagnitude, array: arrayFirstdDay)
    (minValue, maxValue) = getMinMaxValue(minValue: minValue, maxValue:
     maxValue, array: arraySecondDay)

    // take 10 % of the min value to note diference between no values
     and the actual min value
    if minValue == Double.greatestFiniteMagnitude {
        //means that there is nothing to draw
        minValue = 0
        maxValue = 0
    } else {
        // scalle is better if starts always from zero
        minValue = 0
    }


    if arrayFirstdDay.isEmpty {
        arrayOfDataFirst = Array(repeating: 0, count: 24)
    } else {
        arrayOfDataFirst = getArrayOfDataFromSelectedComponent(array:
         arrayFirstdDay, minValue: minValue)
    }

    if arraySecondDay.isEmpty {
        arrayOfDataSecond = Array(repeating: 0, count: 24)
    } else {
        arrayOfDataSecond = getArrayOfDataFromSelectedComponent(array:
         arraySecondDay, minValue: minValue)
    }

    arrayOfCrimeaRoseData = [CrimeaRoseData]()

    arrayOfCrimeaRoseData.append(CrimeaRoseData(arrayOfData:
     arrayOfDataFirst, color: firstUIColor))
    arrayOfCrimeaRoseData.append(CrimeaRoseData(arrayOfData:
     arrayOfDataSecond, color: secondUIColor))


    crimeaRoseView.drawRose(arrayOfCrimeaRoseData:
     arrayOfCrimeaRoseData, arrayOfLabels: getArrayOfLabels(), maxValue:
     maxValue, minValue: minValue)

}

///draws the data to compare by week
private func drawChartForWeek(){
```

```
    let arrayFirstWeek = DbConnection().getDateForWeek(currentDate:
     firstDateToCompare)
    let arraySecondWeek = DbConnection().getDateForWeek(currentDate:
     secondDateToCompare)

    var arrayOfDataFirst: [Double]
    var arrayOfDataSecond: [Double]

    var (minValue, maxValue) = getMinMaxValue(minValue:
     Double.greatestFiniteMagnitude, maxValue:
     Double.leastNormalMagnitude, array: arrayFirstWeek)
    (minValue, maxValue) = getMinMaxValue(minValue: minValue, maxValue:
     maxValue, array: arraySecondWeek)

    // take 10 % of the min value to note diference between no values
     and the actual min value
    if minValue == Double.greatestFiniteMagnitude {
        //means that there is nothing to draw
        minValue = 0
        maxValue = 0
    } else {
        // scalle is better if starts always from zero
        minValue = 0
    }


    if arrayFirstWeek.isEmpty {
        arrayOfDataFirst = Array(repeating: 0, count: 7)
    } else {
        arrayOfDataFirst = getArrayOfDataFromSelectedComponent(array:
         arrayFirstWeek, minValue: minValue)
    }

    if arraySecondWeek.isEmpty {
        arrayOfDataSecond = Array(repeating: 0, count: 7)
    } else {
        arrayOfDataSecond = getArrayOfDataFromSelectedComponent(array:
         arraySecondWeek, minValue: minValue)
    }

    arrayOfCrimeaRoseData = [CrimeaRoseData]()

    arrayOfCrimeaRoseData.append(CrimeaRoseData(arrayOfData:
     arrayOfDataFirst, color: firstUIColor))
    arrayOfCrimeaRoseData.append(CrimeaRoseData(arrayOfData:
     arrayOfDataSecond, color: secondUIColor))


    crimeaRoseView.drawRose(arrayOfCrimeaRoseData:
     arrayOfCrimeaRoseData, arrayOfLabels: getArrayOfLabels(), maxValue:
     maxValue, minValue: minValue)

}

///draws the data to compare by month
private func drawChartForMonth(){
    let arrayFirstMonth = DbConnection().getDateForMonth(currentDate:
     firstDateToCompare)
    let arraySecondMonth = DbConnection().getDateForMonth(currentDate:
     secondDateToCompare)

    var arrayOfDataFirst: [Double]
    var arrayOfDataSecond: [Double]

    var (minValue, maxValue) = getMinMaxValue(minValue:
     Double.greatestFiniteMagnitude, maxValue:
     Double.leastNormalMagnitude, array: arrayFirstMonth)
    (minValue, maxValue) = getMinMaxValue(minValue: minValue, maxValue:
     maxValue, array: arraySecondMonth)

    // take 10 % of the min value to note diference between no values
     and the actual min value
    if minValue == Double.greatestFiniteMagnitude {
        //means that there is nothing to draw
        minValue = 0
        maxValue = 0
    } else {
        // scalle is better if starts always from zero
        minValue = 0
    }


    if arrayFirstMonth.isEmpty {
        arrayOfDataFirst = Array(repeating: 0, count: 12)
    } else {
        arrayOfDataFirst = getArrayOfDataFromSelectedComponent(array:
         arrayFirstMonth, minValue: minValue)
    }

    if arraySecondMonth.isEmpty {
        arrayOfDataSecond = Array(repeating: 0, count: 12)
    } else {
        arrayOfDataSecond = getArrayOfDataFromSelectedComponent(array:
         arraySecondMonth, minValue: minValue)
    }

    //array that handles the data
    arrayOfCrimeaRoseData = [CrimeaRoseData]()

    arrayOfCrimeaRoseData.append(CrimeaRoseData(arrayOfData:
     arrayOfDataFirst, color: firstUIColor))
    arrayOfCrimeaRoseData.append(CrimeaRoseData(arrayOfData:
     arrayOfDataSecond, color: secondUIColor))

    crimeaRoseView.drawRose(arrayOfCrimeaRoseData:
     arrayOfCrimeaRoseData, arrayOfLabels: getArrayOfLabels(), maxValue:
     maxValue, minValue: minValue)

}
```

```
///draws the data to compare by year
private func drawChartForYear(){
    let arrayFirstYear = DbConnection().getDateForYear(currentDate:
     firstDateToCompare)
    let arraySecondYear = DbConnection().getDateForYear(currentDate:
     secondDateToCompare)

    var arrayOfDataFirst: [Double]
    var arrayOfDataSecond: [Double]

    var (minValue, maxValue) = getMinMaxValue(minValue:
     Double.greatestFiniteMagnitude, maxValue:
     Double.leastNormalMagnitude, array: arrayFirstYear)
    (minValue, maxValue) = getMinMaxValue(minValue: minValue, maxValue:
     maxValue, array: arraySecondYear)

    // take 10 % of the min value to note diference between no values
     and the actual min value
    if minValue == Double.greatestFiniteMagnitude {
        //means that there is nothing to draw
        minValue = 0
        maxValue = 0
    } else {
        // scalle is better if starts always from zero
        minValue = 0
    }

    if arrayFirstYear.isEmpty {
        arrayOfDataFirst = Array(repeating: minValue, count: 12)
    } else {
        arrayOfDataFirst = getArrayOfDataFromSelectedComponent(array:
         arrayFirstYear, minValue: minValue)
    }

    if arraySecondYear.isEmpty {
        arrayOfDataSecond = Array(repeating: minValue, count: 12)
    } else {
        arrayOfDataSecond = getArrayOfDataFromSelectedComponent(array:
         arraySecondYear, minValue: minValue)
    }


    //array that handles the data
    arrayOfCrimeaRoseData = [CrimeaRoseData]()

    arrayOfCrimeaRoseData.append(CrimeaRoseData(arrayOfData:
     arrayOfDataFirst, color: firstUIColor))
    arrayOfCrimeaRoseData.append(CrimeaRoseData(arrayOfData:
     arrayOfDataSecond, color: secondUIColor))


    crimeaRoseView.drawRose(arrayOfCrimeaRoseData:
     arrayOfCrimeaRoseData, arrayOfLabels: getArrayOfLabels(), maxValue:
     maxValue, minValue: minValue)
}


/// returns an array based on the component selected in the
 segmentedControl
private func getArrayOfDataFromSelectedComponent(array: [DataModel],
 minValue:Double)->[Double]{
    var arrayOfData = [Double]()

    for obj in array {
        if obj.isNull {
            arrayOfData.append(minValue)
        } else {
            switch segmentedControl.selectedSegmentIndex {
                case 0:
                    arrayOfData.append(obj.temperature)
                    break
                case 1:
                    arrayOfData.append(obj.depth)
                    break
                case 2:
                    arrayOfData.append(obj.pressure)
                    break
                case 3:
                    arrayOfData.append(obj.turbidity)
                    break
                case 4:
                    arrayOfData.append(obj.ph)
                    break
                case 5:
                    arrayOfData.append(obj.oxygen)
                    break
                default:
                    fatalError("segmented control index error in
                     CrimeaRoseViewController")
            }
        }
    }

    return arrayOfData
}

private func getMinMaxValue(minValue: Double, maxValue: Double, array:
 [DataModel]) -> (Double, Double) {
    var min = minValue
    var max = maxValue
    for obj in array {
        if !obj.isNull {
            switch segmentedControl.selectedSegmentIndex {
            case 0:
                if min > obj.temperature{
                    min = obj.temperature
                }
                if max < obj.temperature{
                    max = obj.temperature
```

```
                }
                break
            case 1:
                if min > obj.depth{
                    min = obj.depth
                }
                if max < obj.depth{
                    max = obj.depth
                }
                break
            case 2:
                if min > obj.pressure{
                    min = obj.pressure
                }
                if max < obj.pressure{
                    max = obj.pressure
                }
                break
            case 3:
                if min > obj.turbidity{
                    min = obj.turbidity
                }
                if max < obj.turbidity{
                    max = obj.turbidity
                }
                break
            case 4:
                if min > obj.ph{
                    min = obj.ph
                }
                if max < obj.ph{
                    max = obj.ph
                }
                break
            case 5:
                if min > obj.oxygen{
                    min = obj.oxygen
                }
                if max < obj.oxygen{
                    max = obj.oxygen
                }
                break
            default:
                fatalError("segmented control index error in
                 CrimeaRoseViewController")
            }
        }
    }
    return (min,max)
}

/// returns an array of the labels for the selected data
private func getArrayOfLabels() -> [String]{
    var arrayOfLabels = [String]()
    switch dateTypePicker.selectedRow(inComponent: 0) {
        case 0: //Day
            for i in 1...24 {
                arrayOfLabels.append(String(i))
            }
            break
        case 1: //Week
            for i in 0..<7 {

                 arrayOfLabels.append(Calendar.current.shortWeekdaySymbo
                 ls[i])
            }
            break
        case 2: //Month
            var range = Calendar.current.range(of: .day, in: .month,
             for: firstDateToCompare)!
            var numDays = range.count

            range = Calendar.current.range(of: .day, in: .month, for:
             secondDateToCompare)!
            if numDays < range.count {
                numDays = range.count
            }
            for i in 1...numDays {
                arrayOfLabels.append(String(i))
            }
            break
        case 3: //Year
            for i in 0..<12 {

                 arrayOfLabels.append(Calendar.current.shortMonthSymbols
                 [i])
            }
            break
        default:
            fatalError("pickerView in crimeaRose have more items then
             expected")
    }
    arrayLabels = arrayOfLabels
    return arrayOfLabels
}

/// updates the rose by checking what is selected in the dateTypePicker
private func updateRose(){
    switch dateTypePicker.selectedRow(inComponent: 0) {
    case 0: //Day
        drawChartForDay()
        break
    case 1: //Week
        drawChartForWeek()
        break
    case 2: //Month
        drawChartForMonth()
        break
    case 3: //Year
```

```
        drawChartForYear()
        break
    default:
        fatalError("pickerView in crimeaRose have more items then
         expected")
    }
}

/// updates the labels and all the other components
private func updateLabels(){
    switch dateTypePicker.selectedRow(inComponent: 0) {
    case 0: //Day
        leftDateLabel.text = "\(firstDateToCompare.toString(withFormat:
         "dd MMM yyyy"))"
        rightDateLabel.text = "\
         (secondDateToCompare.toString(withFormat: "dd MMM yyyy"))"
        break
    case 1: //Week
        leftDateLabel.text = "\
         (firstDateToCompare.startOfWeek!.toString(withFormat: "dd/
         MMM")) - \(firstDateToCompare.endOfWeek!.toString(withFormat:
         "dd/MMM"))"
        rightDateLabel.text = "\
         (secondDateToCompare.startOfWeek!.toString(withFormat: "dd/
         MMM")) - \(secondDateToCompare.endOfWeek!.toString(withFormat:
         "dd/MMM"))"
        break
    case 2: //Month
        leftDateLabel.text = "\(firstDateToCompare.toString(withFormat:
         "MMM yyyy"))"
        rightDateLabel.text = "\
         (secondDateToCompare.toString(withFormat: "MMM yyyy"))"

        break
    case 3: //Year
        leftDateLabel.text = "\(firstDateToCompare.toString(withFormat:
         "yyyy"))"
        rightDateLabel.text = "\
         (secondDateToCompare.toString(withFormat: "yyyy"))"
        break
    default:
        fatalError("pickerView in crimeaRose have more items then
         expected")
    }
    updateRose()

    if indexTouched != -1 {
        blueLabel.text = String(format: "%.02f \
         (getEndOfStringForDataType())",
         arrayOfCrimeaRoseData[0].arrayOfData[indexTouched])
        yellowLabel.text = String(format: "%.02f \
         (getEndOfStringForDataType())",
         arrayOfCrimeaRoseData[1].arrayOfData[indexTouched])
    }
}


//-------------------- Handling touch events
///recognizer for the tap on a view
@objc func didTap(tapGR: UITapGestureRecognizer){
    switch tapGR.state {
    case .ended:
        for subview in crimeaRoseView.subviews {
            if subview.frame.contains(tapGR.location(in:
             crimeaRoseView)) {
                if subview is UILabel {
                    blueLabel.text = ""
                    yellowLabel.text = ""
                }
            }
        }


        let index = crimeaRoseView.checkTap(point: tapGR.location(in:
         crimeaRoseView))
        if(index != -1){
            indexTouched = index
            blueLabel.text = String(format: "%.02f \
             (getEndOfStringForDataType())",
             arrayOfCrimeaRoseData[0].arrayOfData[index])
            yellowLabel.text = String(format: "%.02f \
             (getEndOfStringForDataType())",
             arrayOfCrimeaRoseData[1].arrayOfData[index])
        }
    default:
        break
    }
}

///returns the end symbol for the type of data selected
private func getEndOfStringForDataType() -> String{
    switch segmentedControl.selectedSegmentIndex {
        case 0:
            return "°C"
        case 1:
            return "m"
        case 2:
            return "bar"
        case 3:
            return ""
        case 4:
            return "pH"
        case 5:
            return "mg/L"
        default:
            fatalError("segmented control index error in
             CrimeaRoseViewController")
        }
}
```

```
    //--------------Handles changes
    ///handles the change on the segmented control, basicly only needs to
     check which dateTypePicker is selected and call the respective method
    @IBAction func segmentedControlChanged(_ sender: UISegmentedControl) {
        updateLabels()
    }

    @IBAction func firstDateButtonTouched(_ sender: UIButton) {
        DatePickerDialog().show( "Pick a Date", doneButtonTitle: "Done",
         cancelButtonTitle: "Cancel", defaultDate:
         firstDateToCompare ,datePickerMode: .date) {
            (date) -> Void in
            if let dt = date {
                self.firstDateToCompare = dt
                self.updateLabels()
            }
        }
    }

    @IBAction func secondDateButtonTouched(_ sender: UIButton) {
        DatePickerDialog().show( "Pick a Date", doneButtonTitle: "Done",
         cancelButtonTitle: "Cancel", defaultDate: secondDateToCompare,
         datePickerMode: .date) {
            (date) -> Void in
            if let dt = date {
                self.secondDateToCompare = dt
                self.updateLabels()
            }
        }
    }
}

extension CrimeaRoseViewController: UIPickerViewDelegate,
 UIPickerViewDataSource {
    /// The number of columns of data
    func numberOfComponents(in pickerView: UIPickerView) -> Int {
        return 1
    }

    /// The number of rows of data
    func pickerView(_ pickerView: UIPickerView, numberOfRowsInComponent
     component: Int) -> Int {
        return pickerData.count
    }

    /// The data to return for the row and component (column) that's being
     passed in
    func pickerView(_ pickerView: UIPickerView, titleForRow row: Int,
     forComponent component: Int) -> String? {
        return pickerData[row]
    }

    /// Handles the selection of a new row
    func pickerView(_ pickerView: UIPickerView, didSelectRow row: Int,
     inComponent component: Int) {
        ///remove label because of possible crash index out of range
        indexTouched = -1
        blueLabel.text = ""
        yellowLabel.text = ""

        updateLabels()
    }
}
//
//  PageViewController.swift
//  WhalesAppIOS
//
//  Created by Pedro Ribeiro on 29/05/2018.
//  Copyright © 2018 Pedro Ribeiro. All rights reserved.
//

import UIKit


class PageViewController: UIPageViewController,
 UIPageViewControllerDataSource, UIPageViewControllerDelegate {

    ///array with the name of the view controllers so that they can be
     loaded next
    lazy var arrayUIViews: [UIViewController] = {
        return [self.viewControllerInstance(name: "MainPage"),
                self.viewControllerInstance(name: "CrimeaRose"),
                self.viewControllerInstance(name: "Cubijsm"),
                self.viewControllerInstance(name: "Gps")]
    }()

    private let previousButton: UIButton = {
        let button = UIButton(type: .system)
        button.setTitle("PREV", for: .normal)
        button.translatesAutoresizingMaskIntoConstraints = false
        button.titleLabel?.font = UIFont.boldSystemFont(ofSize: 14)
        button.setTitleColor(.gray, for: .normal)
        button.addTarget(self, action: #selector(prevButtonClicked),
         for: .touchUpInside)
        return button
    }()

    private let nextButton: UIButton = {
        let button = UIButton(type: .system)
        button.setTitle("NEXT", for: .normal)
        button.translatesAutoresizingMaskIntoConstraints = false
        button.titleLabel?.font = UIFont.boldSystemFont(ofSize: 14)
        button.setTitleColor(.mainPink, for: .normal)
        button.addTarget(self, action: #selector(nextButtonClicked),
         for: .touchUpInside)
        return button
    }()
```

```
private lazy var pageControl: UIPageControl = {
    let pc = UIPageControl()
    pc.currentPage = 0
    pc.numberOfPages = arrayUIViews.count
    pc.currentPageIndicatorTintColor = .mainPink
    pc.pageIndicatorTintColor = UIColor(red: 249/255, green: 207/255,
     blue: 224/255, alpha: 1)
    return pc
}()

override func viewDidLoad() {
    super.viewDidLoad()

    self.dataSource = self
    self.delegate = self

    if let firstViewController = arrayUIViews.first {
        setViewControllers([firstViewController], direction: .forward,
         animated: true, completion: nil)
    }

    setupBottomControls()
}

/// for layout of the scrollView
override func viewDidLayoutSubviews() {
    super.viewDidLayoutSubviews()
    for view in self.view.subviews{
        if view is UIScrollView {
            view.frame = UIScreen.main.bounds
        } else if view is UIPageControl {
            view.backgroundColor = .clear
        }
    }
}

/// handles the change on the PageViewController
func pageViewController(_ pageViewController: UIPageViewController,
 viewControllerBefore viewController: UIViewController) ->
 UIViewController? {
    guard let viewControllerIndex = arrayUIViews.index(of:
     viewController) else {
        return nil
    }

    let previousIndex = viewControllerIndex - 1

    guard previousIndex >= 0 else {
        return nil
    }

    guard  arrayUIViews.count > previousIndex else {
        return nil
    }

    return arrayUIViews[previousIndex]
}

/// handles the change on the PageViewController
func pageViewController(_ pageViewController: UIPageViewController,
 viewControllerAfter viewController: UIViewController) ->
 UIViewController? {
    guard let viewControllerIndex = arrayUIViews.index(of:
     viewController) else {
        return nil
    }

    let nextIndex = viewControllerIndex + 1

    guard nextIndex < arrayUIViews.count else {
        return nil
    }

    guard  arrayUIViews.count > nextIndex else {
        return nil
    }

    return arrayUIViews[nextIndex]
}

/// handles the change on the PageViewController
func pageViewController(_ pageViewController: UIPageViewController,
 didFinishAnimating finished: Bool, previousViewControllers:
 [UIViewController], transitionCompleted completed: Bool) {
    let pageContentViewController = pageViewController.viewControllers!
     [0]
    pageControl.currentPage = arrayUIViews.index(of:
     pageContentViewController)!
}


// Private Methods

/// instantiates the views
private func viewControllerInstance(name:String) -> UIViewController {
    return UIStoryboard(name: "Main", bundle:
     nil).instantiateViewController(withIdentifier: name)
}

/// sets the bottom controls and their layouts
private func setupBottomControls() {

    let bottomControlsStackView = UIStackView(arrangedSubviews:
     [previousButton, pageControl, nextButton])

    bottomControlsStackView.translatesAutoresizingMaskIntoConstraints =
     false
    bottomControlsStackView.distribution = .fillEqually

    view.addSubview(bottomControlsStackView)
```

```
        NSLayoutConstraint.activate([
            bottomControlsStackView.bottomAnchor.constraint(equalTo:
             view.safeAreaLayoutGuide.bottomAnchor),
            bottomControlsStackView.leadingAnchor.constraint(equalTo:
             view.safeAreaLayoutGuide.leadingAnchor),
            bottomControlsStackView.trailingAnchor.constraint(equalTo:
             view.safeAreaLayoutGuide.trailingAnchor),
            bottomControlsStackView.heightAnchor.constraint(equalToConstant:
             50)
            ])
    }

    ///handles the touch event on the prev button
    @objc private func prevButtonClicked(){
        // prevents from having negative index
        let nextIndex = max(pageControl.currentPage - 1, 0)
        pageControl.currentPage = nextIndex

        setViewControllers([arrayUIViews[nextIndex]], direction: .reverse,
         animated: true, completion: nil)
    }

    ///handles the touch event on the next button
    @objc private func nextButtonClicked(){
        // prevents from having negative index
        let nextIndex = min(pageControl.currentPage + 1, arrayUIViews.count
         - 1)
        pageControl.currentPage = nextIndex

        setViewControllers([arrayUIViews[nextIndex]], direction: .forward,
         animated: true, completion: nil)
    }
}
//
//  ViewController.swift
//  WhalesAppIOS
//
//  Created by Pedro Ribeiro on 04/05/2018.
//  Copyright © 2018 Pedro Ribeiro. All rights reserved.
//

import UIKit
import RealmSwift

class MainScreenViewController: UIViewController {

    @IBOutlet weak var mainLabel: UILabel!

    @IBOutlet weak var scrollView: UIScrollView!

    private let refreshControl : UIRefreshControl = {
        let refreshControl = UIRefreshControl()
        refreshControl.addTarget(self, action:
         #selector(refreshingScreen(_:)), for: UIControlEvents.valueChanged)

        return refreshControl
    }()

    override func viewDidLoad() {
        super.viewDidLoad()
        // Do any additional setup after loading the view, typically from a
         nib.

        scrollView.refreshControl = refreshControl

        ConnectionToServer().sendDataToServer()
        ConnectionToServer().getDataFromServer(completion: {
            self.setupLabel()
        })

        setupLabel()
    }

    override func didReceiveMemoryWarning() {
        super.didReceiveMemoryWarning()
        // Dispose of any resources that can be recreated.
    }


    @IBAction func connectToDevice(_ sender: UIButton) {
        print("button touched")
        ConnectionToDevice().getData(i: 0)
    }

    func setupLabel(){
        let data = DbConnection().getDataFromDb()

        mainLabel.numberOfLines = 0
        if data.count > 0 {
            mainLabel.text = "Start swiping to explore!"
        } else {
            mainLabel.text = "Currently there is no data to view\n\nPull to
             refresh"
        }
    }

    @objc func refreshingScreen(_ refreshControl: UIRefreshControl){
        print("here")

        ConnectionToServer().sendDataToServer()

        ConnectionToServer().getDataFromServer(completion: {
            self.setupLabel()
            self.refreshControl.endRefreshing()
        })
    }
}
```

```swift
//
//  CrimeaRoseView.swift
//  WhalesAppIOS
//
//  Created by Pedro Ribeiro on 04/07/2018.
//  Copyright © 2018 Pedro Ribeiro. All rights reserved.
//

import UIKit

class CrimeaRoseView: UIView {
    //------------------- private vars cannot be set by the user

    private var arrayOfCrimeaRoseData = [CrimeaRoseData]()
    private var arrayOfLabels = [String]()
    private var arrayOfPaths = [UIBezierPath: Int]()

    //------------------- public vars can be set by the user

    ///color of the line that draws the bounds of the rose default: black
    public var lineColor = UIColor.black

    ///with of the line that draws the bounds of the rose default: 1.0
    public var lineWidth:CGFloat = 1.0

    ///maximum value to show
    public var maxValue:CGFloat = CGFloat.leastNormalMagnitude

    ///minimum value to show
    public var minValue:CGFloat = CGFloat.greatestFiniteMagnitude

    ///font for the labels around the circle -> default "Helvetica-Bold"
     size: 12
    public var labelFont = UIFont(name: "Helvetica-Bold", size: 12)

    ///color for the labels around the circle -> default black
    public var labelTextColor = UIColor.black

    ///hides the labels around the circles -> default false
    private var areLabelsHidden = false


    //----------------------- Public Methods ------------------------//

    /// calls the method to draw the vew with the data. Needs the values,
     the labels, and the colors
    public func drawRose(arrayOfCrimeaRoseData: [CrimeaRoseData],
     arrayOfLabels:[String], maxValue: Double, minValue: Double) {
        //cleans the data
        self.arrayOfCrimeaRoseData = [CrimeaRoseData]()
        self.arrayOfLabels = [String]()
        self.arrayOfPaths = [UIBezierPath: Int]()
        self.maxValue = CGFloat(maxValue)
        self.minValue = CGFloat(minValue)
        _ = self.subviews.map({$0.removeFromSuperview()})

        self.arrayOfCrimeaRoseData = arrayOfCrimeaRoseData
        self.arrayOfLabels = arrayOfLabels

        updateDisplay()
    }


    //----------------- Main Drawing function -------------//

    // An empty implementation adversely affects performance during
     animation.
    override func draw(_ rect: CGRect) {
        //if there is no data to show just present a label saying that
        if(maxValue == minValue) {
            let label = UILabel()

            let x: CGFloat = bounds.midX
            let y: CGFloat = bounds.midY

            label.font = labelFont
            label.frame = CGRect(x: x, y: y, width: bounds.width, height:
             50)
            label.text = "There is no data for the current selected dates"
            label.textAlignment = .center
            label.textColor = labelTextColor
            label.isHidden = areLabelsHidden

            label.center = CGPoint(x: x, y: y)

            self.addSubview(label)
        } else {

            var arrayOfObjs = [ObjtoDraw]()

            for array in arrayOfCrimeaRoseData {

                //seting the angle of each small circle
                let angle = (2*CGFloat.pi) / CGFloat(arrayOfLabels.count)
                var startAngle = CGFloat(3*CGFloat.pi/2);

                for i in 0..<array.arrayOfData.count{
                    let radius = scale(value: CGFloat(array.arrayOfData[i]))

                    arrayOfObjs.append(ObjtoDraw(radius: radius, startAngle:
                     startAngle, endAngle: startAngle + angle, fillColor:
                     array.color, category: i))

                    startAngle = startAngle + angle
                }
            }

            for obj in arrayOfObjs.sorted(by: {$0.radius > $1.radius}){
```

```
            drawSmallSemiCircle(radius: obj.radius, startAngle:
             obj.startAngle, endAngle: obj.endAngle, fillColor:
             obj.fillColor, category: obj.category)
        }

        // draws labels
        let angle = (2*CGFloat.pi) / CGFloat(arrayOfLabels.count)
        var startAngle = CGFloat(3*CGFloat.pi/2);
        for i in 0..<arrayOfLabels.count {
            let obj = arrayOfObjs.filter({$0.startAngle ==
             startAngle}).max(by: {$0.radius < $1.radius})!

            drawLabels(string: arrayOfLabels[i], angle: startAngle +
             (angle/2), radius: obj.radius)

            startAngle = startAngle + angle
        }
    }

}


//------------------- Handling Events ---------------//

/// handles the pinch to zoom gesture -> just scales the view
@objc func didPinch(pinchGR: UIPinchGestureRecognizer){
    switch pinchGR.state {
    case .changed, .ended:
        let scale = pinchGR.scale
        self.transform = self.transform.scaledBy(x: scale, y: scale)
        pinchGR.scale = 1.0
    default:
        break
    }
}

/// handles the pan gesture -> just changes the center of the view to
 the new coordinates
@objc func didPan(panGR: UIPanGestureRecognizer){
    switch panGR.state {
    case .changed, .ended:
        let translation = panGR.translation(in: self)

        self.center.x += translation.x
        self.center.y += translation.y

        panGR.setTranslation(CGPoint(x: 0, y: 0), in: self)


    default:
        break
    }
}
```

```
/// handles the rotation of the view -> does a transformation on the
 view based on the rotation aplied
@objc func didRotate(rotationGR: UIRotationGestureRecognizer){
    switch rotationGR.state {
    case .changed, .ended :
        let rotation = rotationGR.rotation
        self.transform = self.transform.rotated(by: rotation)
        rotationGR.rotation = 0.0
    default:
        break
    }
}

public func checkTap(point: CGPoint)->Int{
    for (path, index) in arrayOfPaths {
        if path.contains(point){
            return index
        }
    }

    return -1
}

//------------------- Methods ----------------------//



/// updates the view call this when there is changes in the data
private func updateDisplay(){
    self.setNeedsDisplay()
    self.setNeedsLayout()
}


///draws a semicircle in the center of the view
private func drawSmallSemiCircle(radius: CGFloat, startAngle: CGFloat,
 endAngle: CGFloat, fillColor: UIColor, category: Int) {

    let path = UIBezierPath()

    path.addArc(withCenter: CGPoint(x: bounds.midX, y: bounds.midY),
     radius: radius, startAngle: startAngle, endAngle: endAngle,
     clockwise: true)
    path.addLine(to: CGPoint(x: bounds.midX, y: bounds.midY))
    path.close()

    path.lineWidth = lineWidth

    lineColor.setStroke()
    fillColor.setFill()
    path.fill(with: .overlay, alpha: 0.8)
    path.stroke()

    arrayOfPaths[path] = category
}
```

```
    ///https://stackoverflow.com/questions/5294955/how-to-scale-down-a-
     range-of-numbers-with-a-known-min-and-max-value ---
    /// a is 0 and b is the viewRadius
    private func scale(value: CGFloat) -> CGFloat{
        return (viewRadius * (value - minValue)) / (maxValue - minValue)
    }


    /// draws the text Labels -> label is centered 15 points away from the
     maximum radius value obtained
    private func drawLabels(string: String, angle: CGFloat, radius: CGFloat)
     {
        let label = UILabel()

        var r: CGFloat

        if radius > viewRadius/2 {
            r = radius
        } else {
            r = viewRadius/2
        }

        let x: CGFloat = bounds.midX + ((r + 15) * cos(angle))
        let y: CGFloat = bounds.midY + ((r + 15) * sin(angle))

        label.font = labelFont
        label.frame = CGRect(x: x, y: y, width: 30, height: 30)
        label.text = string
        label.textAlignment = .center
        label.textColor = labelTextColor
        label.isHidden = areLabelsHidden

        label.center = CGPoint(x: x, y: y)

        self.addSubview(label)
    }




}

extension CrimeaRoseView {

    private var viewRadius: CGFloat {
        get {
            if bounds.width > bounds.height {
                return (bounds.height - 50)/2
            } else {
                return (bounds.width - 50)/2
            }
        }
    }

    private var getRangeForRadius: CGFloat {
        get {
            return abs(maxValue) + abs(minValue)
        }
    }


}

struct CrimeaRoseData {
    var arrayOfData = [Double]()
    var color = UIColor()

    init(arrayOfData: [Double], color: UIColor) {
        self.arrayOfData = arrayOfData
        self.color = color
    }
}

struct ObjtoDraw {
    var radius: CGFloat = 0
    var startAngle: CGFloat = 0
    var endAngle: CGFloat = 0
    var fillColor = UIColor()
    var category = -1;
    init(radius: CGFloat, startAngle: CGFloat, endAngle: CGFloat, fillColor:
     UIColor, category: Int) {
        self.radius = radius
        self.startAngle = startAngle
        self.endAngle = endAngle
        self.fillColor = fillColor
        self.category = category
    }
}

//
//  PopupCell.swift
//  WhalesAppIOS
//
//  Created by Pedro Ribeiro on 07/08/2018.
//  Copyright © 2018 Pedro Ribeiro. All rights reserved.
//

import UIKit

class PopupCell: UICollectionViewCell {

    var cellInfo: CellInfo? {
        didSet {
            textLabel.text = cellInfo?.text
            iconImageView.image = UIImage(named: (cellInfo?.iconName)!)
        }
    }

    let textLabel: UILabel = {
```

```
        let label = UILabel()
        label.text = "ups"
        return label
    }()

    let iconImageView: UIImageView = {
        let imageView = UIImageView()
        imageView.image = UIImage(named: "play_button")
        imageView.contentMode = .scaleAspectFill
        imageView.tintColor = .black
        return imageView
    }()


    override init(frame: CGRect) {
        super.init(frame: frame)

        setupViews()
    }

    required init?(coder aDecoder: NSCoder) {
        fatalError("init(coder:) has not been implemented")
    }

    func setupViews(){
        addSubview(textLabel)
        addSubview(iconImageView)

        addConstraintsWithFormat(format: "H:|-10-[v0(30)]-10-[v1]|", views:
         iconImageView, textLabel)
        addConstraintsWithFormat(format: "V:|[v0]|", views: textLabel)
        addConstraintsWithFormat(format: "V:[v0(30)]", views: iconImageView)

        addConstraint(NSLayoutConstraint(item: iconImageView,
         attribute: .centerY , relatedBy: .equal, toItem: self,
         attribute: .centerY, multiplier: 1, constant: 0))
    }

}
```