INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO

MESTRADO EM ENGENHARIA COMPUTAÇÃO E INSTRUMENTAÇÃO MÉDICA

Otimização do processo de monitorização de elétrodos seletivos de ião

DANIEL JOÃO CERQUEIRA DA COSTA

Setembro de 2019

POLITÉCNICO
DO PORTO

Ion-selective electrode research automation

BioMark Research Process Optimization

Daniel João Cerqueira da Costa

1101325@isep.ipp.pt

Research Thesis
October 8, 2019

With orientation of:
PhD Goreti Sales
Physics Department and BioMark Laboratory

In partial fulfillment of the requirements for the degree of Master of Science in Computing and Medical Instrumentation Engineering

Instituto Superior de Engenharia do Porto
Porto, Portugal

Every morning we are born again. What we do today is what matters most.   Buddha

Acknowledgments

In 2013, I started this project, but, due to personal/familiar affairs, I had to interrupt. I restarted in 2018 due to the support of my supervisor Goreti Sales. For this reason, for believing in me and this project, and for the availability, I would like to special thanks her.

For the lack of family time this year, and trying to be an inspiration to believe in finishing the chapters of our lives, I would like to thank my beautiful wife and two daughters for their patience, support and love.

For the true motivation of this project, I would like to thank my mother and sister, who together with me have formed a supportive and loving force in a losing battle against my late father's cancer.

Finally, I would like to thank the Biomark team for their availability, tireless search for cables and contacts with suppliers, for the time invested with me in discovering the functionality of the equipments, in particular Ricardo Castro, Felismina Moreira, Liliana Truta and Ana Moreira.

Abstract

During the Ion-selective electrode research process, there are a lot of manual calibrations needed in order to achieve better and more effective results. The main goal of most of the researchers in these areas is about the material behaviours and reactions, but there is no way to test them without building the electrode and testing against various possibilities and mediums.

Due to this, the research process usually cannot be a continuous process, since there are some interruptions for the manual process to validate all the previous developments. The human based calibration process is fundamental for many conclusions, but, most of the calibrations logic can be implemented in a machine that automates the process, collects the data, and generates the necessary output.

The main goal of this MSc thesis is to optimize the calibration process building a calibration box that is fully configurable, and can be replicated, replacing the researcher manual steps during the membrane testing. Herein, it is possible to find the selected hardware and software architecture used, and also the business logic implemented to achieve the automation of these process. The architecture and implementation were designed to be able to work with a digital potentiometer (Crison GLP 21 pH Potentiometer) and a digital precise fluid pump (Legato 100).

The solution contains four main components:

- External Devices (Digital pH Meter, Digital Peristaltic Pump);
- Server side web services and web application;
- Cloud Based Deployment (Serverless, Storage, Database);
- Hardware automation box;

The implementation of this thesis uses the following technologies:

- Code: Java, Spring Boot, Spring Data;
- Data: JSON, Excel XML OpenFormat, PostgresSQL;
- Hardware: RaspberryPi, Relay Boards, LCD Display, Button, LED;
- I/O: Ethernet, USB to RS232, RCA, BNC;
- Cloud: Heroku Serverless; Heroku PostgresSQL; Amazon S3 Storage;

All the membranes used were made by BioMark researchers.

Keywords: Automation, Java, Research, Spring, Data, Raspberry Pi, GPIO, Cloud, Serverless, PostgreSQL, S3, BioMark

Resumo

Durante o processo de pesquisa de eletrodos seletivos de íons, há muitas calibrações manuais necessárias para obter resultados melhores e mais eficazes. O objetivo principal da maioria dos investigadores nestas áreas é relativa aos comportamentos e reações dos materiais, mas não existe uma maneira de testá-los sem construir o eletrodo e testar contra várias possibilidades e meios.

Por este motivo, o processo de pesquisa geralmente não pode ser um processo contínuo, uma vez que existem algumas interrupções no mesmo para manualmente validar todos os desenvolvimentos anteriores. O processo de calibração baseado em trabalho manual é fundamental para uma grande parte das conclusões, mas a maior parte da lógica de calibração pode ser implementada e executada por uma máquina que automatiza o processo, recolhe os dados e gera a saída necessária para posterior analise do investigador.

O objetivo principal desta tese de mestrado é otimizar o processo de calibração construindo uma caixa de calibração totalmente configurável e que pode ser replicada, substituindo as etapas manuais do investigador durante o teste das membranas. Neste relatório, é possível encontrar a arquitetura de hardware e software selecionada e utilizada bem como a lógica implementada para alcançar a automação desses processos. A arquitetura e a implementação foram projetadas para poder trabalhar com um potenciômetro digital (Potenciômetro de pH Crison GLP 21) e uma bomba de fluido (Legato 100).

A solução contém quatro componentes principais:

- Dispositivos Externos (Medidor de pH Digital, Bomba Peristáltica Digital);
- *Server side web services* e aplicação Web;
- *Cloud Based Deployment (Serverless, Storage, Database)*;

- Caixa de automação (*Hardware*);

A implementação desta tese utiliza as seguintes tecnologias:

- Código: *Java, Spring Boot, Spring Data*;
- Dados: *JSON, Excel XML OpenFormat, PostgresSQL*;
- Hardware: RaspberryPi, Relay Boards, LCD Display, Button, LED;
- *I/O: Ethernet, USB to RS232, RCA, BNC*;
- *Cloud: Heroku Serverless; Heroku PostgresSQL; Amazon S3 Storage*;

Todas as membranas utilizadas foram desenvolvidas por investigadores da BioMark.

Keywords: Automation, Java, Research, Spring, Data, Raspberry Pi, GPIO, Cloud, Serverless, PostgreSQL, S3, BioMark

Contents

List of Figures

List of Tables

Abbreviations

URL	Uniform Resource Locator
CPU	Central Processing Unit
CRUD	Create, Read, Update, Delete
JVM	Java Virtual Machine
CI	Continuous Integration
CD	Continuous Delivery
ISEP	Instituto Superior de Engenharia do Porto
API	Application Programming Interface
MIT	Massachusetts Institute of Technology
HTML	Hypertext Markup Language
XML	Extended Markup Language
CSS	Cascading Style Sheets
JS	JavaScript
GPIO	General Purpose Input/Output
mV	Milivolt
JSON	JavaScript Object Notation
REST	Representational State Transfer
MVC	Model View Controller
UI	User Interface
UX	User Expririence
POJO	Plain Old Java Object
DTO	Data Transfer Object
CB	Calibration Box
MVP	Minimum Viable Product
QMP	Quality Management Process
ISE	Ion-Selective Electrodes
HBV	Hepatitis B Virus
HPV	Human Papillomavirus
WHO	World Health Organization
USB	Universal Serial Bus
LCD	Liquid Cristal Display
LED	Light Emitting Diode
BNC	Bayonet Neill–Concelman

Chapter 1

Introduction

1.1 Cancer

Cancer has killed 9.8 million people worldwide, in 2018, according to the World Health Organization (WHO). However, and also according to WHO, a range of 30% to 50% of these deaths could be prevented by healthy life style or by immunization against cancer causing infections (HBV, HPV), and the early detection can be the key for the remaining cases. The latest World Cancer Report (published in 2014 by WHO) reports that the leading causes of cancer death are due to lung, liver, stomach, colorectal and breast cancer. [1, 2]

There are over 100 types of cancer, and each part of the body can be affected. It all starts with the genetic mutation of a cell, a phenomenon called carcinogenesis that consists of altering a cell to make it abnormal, and its uncontrolled proliferation that eventually invades nearby tissues. This invasion process is enhanced by the ability of these cells to release growth factors and digestive enzymes that promote their continued division. This process is composed of the stages (three stages) of initiation, promotion and progression of the tumor. The process of initiation (stage 0 to 1) is rapid and irreversible, and begins with the genetic mutation of a normal cell, which after proliferates to create a population of tumor cells. Next comes a long process of tumor promotion (stage 2 and 3), either as a result of the proliferation of the tumor cells or of their involvement. In the last stage (stage 4), the progression happens with the growth of the tumor, which may have a metastatic and invasive nature. [3, 4]

1.2 Tumor Biomarkers

Early cancer diagnosis is the key tool to improve survival rates. According to the *Cancer treatment and survivorship statistics, 2014*, the wide coverage and dissemination of mammography, as an early screening tool resulted in a 10 and 15 year survival rate of 83.1% and 77.8%, respectively. [5] But even with advances in preventive methods, it is usually difficult to detect cancer, since morphological changes in tissues are not always evident, especially in the first stage of the disease. The most common and accurate methods for early detection of the disease are currently based on invasive biopsy procedures or exploratory surgeries that entail greater risks and costs for patients.

For these reasons, the discovery of tumor markers has emerged as an important discovery for early detection of cancer, which in turn assists in monitoring the patient's response during treatments and also allows early analysis in healthy individuals of their cancer risk. [1, 2] These markers are present in tumour tissues or serum, and include DNA, mRNA, enzymes, metabolites, transcription factors, and cell surface receptors. Cancer biomarkers are produced by the body in response to or during cancer growth and can easily be included in a screening process as they can be detected in biological samples (urine, saliva, blood, tissues).[2]

In order to avoid false positives, criteria were set for considering a tumor marker, such as how specific and sensitive the marker is to a particular tumor. It is equally important to effectively correlate marker levels with the respective tumor stage [6, 7]. Numerous tumor identifiers are currently identified, such as CEA, CA15-3, CA125, CA19-9, PSA, among others. With these markers it is already possible to clinically monitor a wide range of cancer types, such as colorectal, breast, ovarian, uterus, lung, pancreas, liver, prostate, among others [2, 8].

Thus, being able to detect and monitor in an early stage such tumour markers would represent a low-cost, low-intervention form of diagnosis that could further be exploited to provide rapid, low-cost screening solutions, allowing early diagnosis to yield higher rates of cancer survival [1, 9]. This leads to the need of developing new sensors and devices that allow the detection of such biomarkers, especially in point-of-care. Different approaches may be employed to this end, including ion-selective electrodes.

1.3 Ion-selective electrodes

Among the possible means of detecting biomarkers are Ion-Selective Electrodes (ISEs) which are already widely used in various fields of analysis [10, 11, 12]. Precision and speed, as well as its cost-effectiveness and sensitivity to different concentrations are found in the numerous reasons why researchers are exploring this methodology of analysis. [13] Moreover, ISEs may allow direct sample readings, avoiding tedious and expensive pre-treating stages before analysis itself. Short response times, in the order of seconds, also make ISEs appropriate devices for analytical control, most especially when portability and low-cost are demanding features, as required for cancer biomarker screening programs.

In general, the construction and manipulation of ISE devices is an expeditious and low-cost approach if solid contact configurations are used. An example of these may be seen in figure 1.1, describing conventional and tubular-shape electrodes [14]. In these, a Perspex body is used to support a graphite-based electrical contact that shall support a suitable selective membrane. This selective membrane is the core element of the device and contains a specific composition that allows targeting in a selective manner any compound that displays an ionic charge. This is the case of many cancer biomarkers in liquid biopsies, which are mostly charged biochemical compounds circulating in the blood.

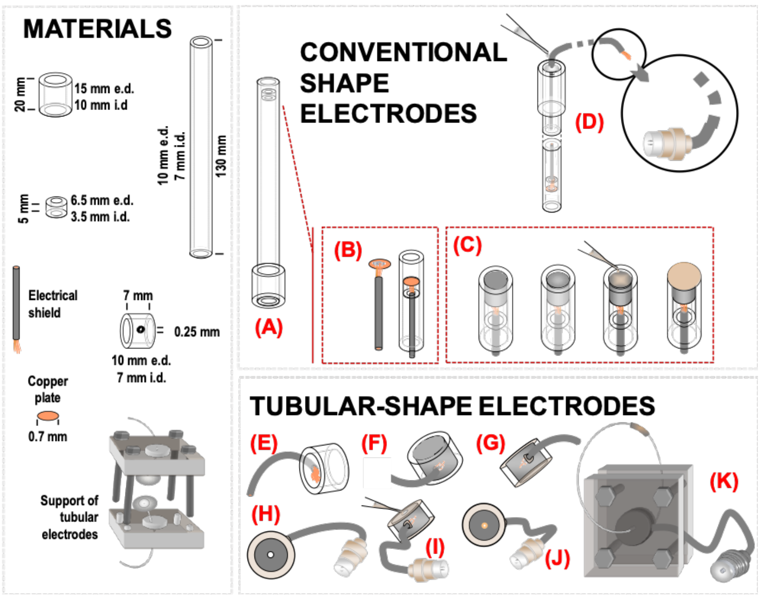


Figure 1.1: Construction of conventional and tubular shape SDZ selective electrodes.

Construction of conventional and tubular shape SDZ selective electrodes (figure 1.1) made of Perspex® tubes with a shielded electrical cable using a copper plate as electrical contact to a graphite-based conductive support. A: electrode body with mounted tubes; B: electrical connection to the copper plate placed in the electrode body; C: fixation of electrical cable and connection to a BNC antenna connector; D: graphite-epoxy conductive support with a drilled cavity ($\approx$ 1,0 mm depth) and membrane applied over it; E and F: inner cavity filled with graphite-epoxy conductive matrix that fixes the electrical cable inside; G: polished and isolated external surfaces; H: internal hole drilled ($\approx$ 1,0 mm Φ); I: application of selective membrane; J: membrane attached to the walls of the inner hole; K: tubular electrode placed in a flow support (closed circuit for membrane conditioning). As shown in [14].

1.4 Objectives and Motivation

The signal produced by solid-contact ISEs is a potential difference (typically expressed in mV), measured between it and a reference electrode that are both dipped in the same solution and connected to the same potentiometer. This potential difference changes in a programmed way when a target compound is present in a given concentration, which is established by a calibration procedure. A calibration is typically made in several stages. First, the ISE needs to stabilize in a background medium solution. Next, a known amount of target analyte solution is added into this background medium, the resulting solution is homogenised, and the potential recorded in the potentiometer is noted after stabilization. This addition procedure is repeated for known amounts of target analyte in order to generate increasing concentrations (amounts) of such analyte in the solution where the electrodes stand (Figure 1.2, A). The several potential differences recorded for each concentration are plotted after, using the log concentration of the target analyte at each moment in x axis and the potential difference in y axis. This plot allows extracting all analytical parameters related to the performance of an ISE.

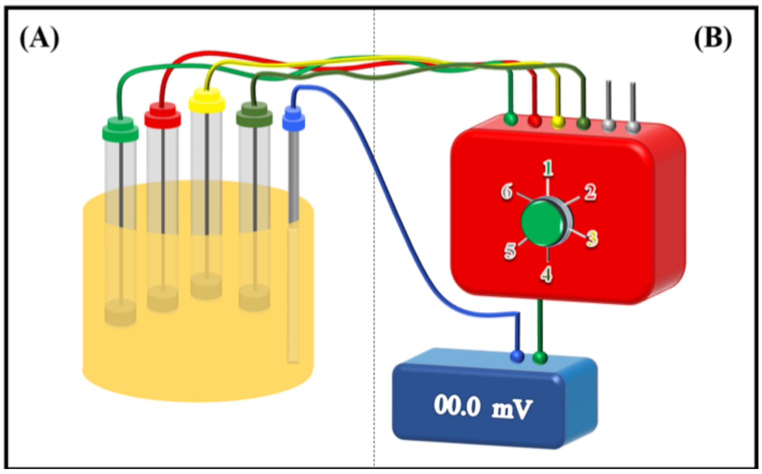


Figure 1.2: A potentiometric cell (A) combining 4 ISEs and 1 reference electrode, connected to the (B) the multi commutation point and the potentiometer.

The composition of the selective membrane casted on the conductive graphite support is the main critical aspect behind the good operation of ISEs. There is no specific formulation to generate a given ISE for a specific target marker, and therefore its composition is defined after intense optimization processes, established by trial/error approaches. To this end, different compounds are tested, in different percentages and for replicate units (a minimum of three electrodes for three assays is required). The unique tool that is currently available to help the optimization of the membrane composition is a home-made multi-channel reading box manually controlled. The reading box used by BioMark sensor research group allows reading 6 electrodes in the same solution condition (Figure 1.2, B). To this end, the response of each ISE is read manually and individually. In brief, one electrode is read by the operator in a specific channel, after this reading the operator switches manually the commutation point to another channel, to collect the potential signal from another electrode. Ensuring that the potential signals generated by each electrode at the time of reading is correct is therefore a hard task and requires a highly experienced operator. Other approach nearing an automated reading of ISEs include the collection of the potential signal by a computer, but these are made only for a single electrode along the same calibration procedure (only 1 electrode may be calibrated each time). Thus, the main purpose of this work is to develop a multi-commutation point unit that may collect the signals of multiple ISEs is an automated manner, while also allowing following automatically the complete experimental calibration procedure, with minimal or none operator intervention.

1.5 The Problem, Contributions and Strategy

In order to achieve the interaction with the existing equipment's using the available interfaces, the first approach was to search about any possible integration software provided by the equipment's vendors. The equipment's interfaces are common standard interfaces, since Crison pH meter has the RS232 interface, and the Legato 100 pump has the USB interface with a proprietary driver. Regarding the Crison device, there are common configurations on the documentation related with the serial port communication. The selected process to achieve this project success was inspired and adapted based in the ISO 9000,9001 seven Quality Management Process (QMPs)[15] principles.

With the device interfacing strategy defined, the process logic definition is the next step. There are several points that need to be addressed: understanding the experimental procedure (QMP 4)[15, p.8] and all its logic, what is behind the calibration process in order to define what is needed to create a membrane, what is its target and goals, or in what solution will the electrodes be tested. The strategy used was to follow all the BioMark process regarding electrodes development, by following all the manual procedures made by the BioMark researchers.

After the previous definition, it is time to start with the development process. In order to simplify the continuous testing process, and the BioMark envolvement (QMP

3)[15, p.6-7], the approach consisted in scheduling meetings with delivery's and discussing any necessary adjustment and the further development.

1.6 Technical Requirements

With the limited timeframe, here we have the Minimum Viable Product (MVP) requirements:

- Hardware should be able to read from Crison pH Meter;
- Hardware should be able to operate a precision automatic pump;
- Hardware should be able to be operated manually (without automatic pump);
- The collected data should be available in an excel format (compatible with the current manual analysis process);
- All the calibration processes should be organized and persisted in a database;
- The calibration stability parameters should be configurable;
- The displayed values should be in a scientific notation;
- All the platform should be operated using the English language;

1.7 Future Impact

Improving the research calibration process, the researchers are able to focus what is important, that is build the membranes based on the materials evolution and calibration results. This platform also avoids human error, specially with the stability pattern enforcement. This solution is also able to return the calibration results faster and without human effort.

1.8 Work plan and expected results

A plan has been prepared to keep the work organized and with a common thread, whose timing, taking into account the defined objectives, involved the following steps:

- Introduction and evolvement with the proposed project;
- State of the art study;
- Research on the software and hardware technologies to use;

- Definition of the architecture to be adopted for the system developed;
- Design and develop the server side application;
- Review and adjustments of the server side application with the BioMark team;
- Design and develop the hardware side application;
- Assembly and configure the hardware prototype;
- Beginning of the preparation of this dissertation document;
- System testing and validation;
- Conclusion of the final dissertation document.

The expected result is an Hardware and Software solution that automates all the calibration process.

1.9 Report Organization

In the state of the art phase of this project, there was concluded that there is no solutions on the market able to combine the usage and also the different environments and brands involved in this project. Due to this, this project does not have the state of the art usual chapter. Then, this thesis is divided in six chapters.

In this **first chapter** the thesis is framed, there is an introduction with the problem and concepts, and the objectives are listed and all the work planning is detailed.

The **second chapter** describes the manual and automatic calibration process as a complement for the introduction made in the first chapter;

The **third chapter** is the biggest chapter of this document, and describes in detail all the research made to define the architecture for this project execution, including the hardware, software and the communication between both parts of the solution.

At the **fourth chapter**, the server side solution development is described including all the relevant technical details.

The **fifth chapter** describes in detail the Calibration Box development, including the assembly process, the operating system configuration and the software development. Since this project consist in a solution, in this chapter, particularly in the 5.3 sub section, it is possible to observe the conclusions and demonstration of results.

The last **sixth chapter** contains the conclusions, final considerations and future work proposals.

Chapter 2

Calibration Process

2.1 Manual Process

The manual process starts with the research preparation by calculating the concentration and components in a excel folder.

After that, the researcher starts the calibration, this part consists in the observation of Crison pH meter values regarding each membrane (by manually commuting between them). The researcher waits for the stability point, taking notes in a manual table on a paper or on the computer. The researcher also adjusts the concentration manually during the process as initially planed.

Each membrane can takes up to 2 minutes to stabilise, and should be tested with up to 15 different concentrations. In a worst scenario with 8 different membranes to be calibrated this process can take up to 4 hours to be completed, removing human errors and intervals.

2.2 Automated Process

The intent of this process automation is not to accelerate this process results, since this is physically not possible. Like in the manual process, but removing the human effort and criteria, the automation box integrates with the devices in order to measure the membrane values and also to add some additional fluid to the concentration.

In the figure 2.1, 1 we can observe a 8 channel calibration box interfacing with Crison GLP21 pH meter using a serial port (RS232) (2.1, 6) to read the measured values. This calibration box also commutes automatically between the 8 possible membrane

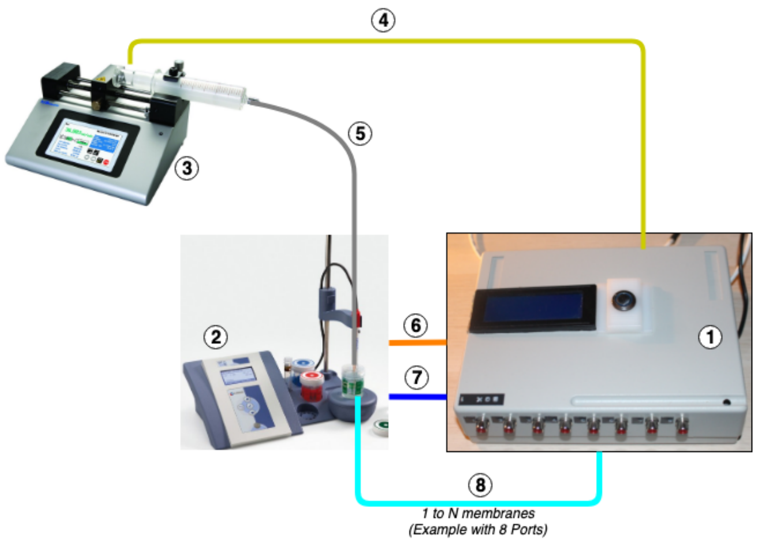


Figure 2.1: Calibration Box interfacing with Crison GPL21 and Legato 100

connections (2.1, 8) interconnecting it with the Crison device via a coaxial connection with a BNC terminator (2.1, 7).

The Calibration Box is able to operate with an automatic pump (Legato 100)(2.1, 3) controlled via USB (2.1, 4). In this scenario the user just have to wait until the end of the process, since the CB do all the work for the researcher, measuring, waiting for the stability point by configured criteria, and adding the necessary concentration (2.1, 5). The user is able to configure if the CB has or not an automatic pump installed. If not, the CB automatically commutes between configured membranes until reach the last measure, after that, the CB instructs the user to add the amount of extra fluid to the cumulative concentration, by displaying it on the display, and waiting for user confirmation by clicking the blinking (LED) button.

Chapter 3

Architecture

From the beginning the main architecture defined was the client-server, since each Calibration Box (CB) is a Client, and then the server consolidates all the configurations and manage the CB status. With this architecture the business logic stays on the server side making it easy to multiply the CB units in order to automate more than one research process at the same time.

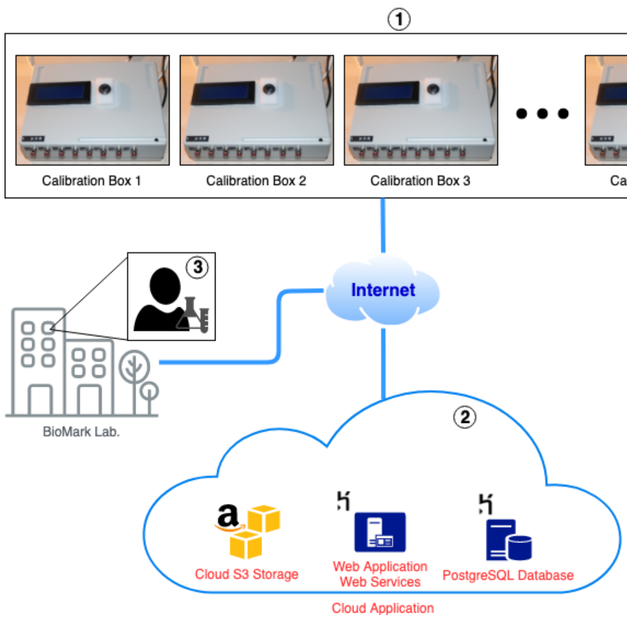


Figure 3.1: A group of Calibration Boxes (1), connected with the web services deployed on the cloud (2), operated in the lab by the researcher (3) via the Internet.

There are two main sub-architectures defined. The server side software and the client CB hardware, both described in detail in the next two sections (3.1, 3.2).

3.1 Software Architecture

On the software architecture, this section is divided in the following parts:

- Server-Side Application Architecture 3.1.1;
- Calibration Box Software Architecture 3.1.2

3.1.1 Server-Side Application Architecture

The server-side web application has two requisites. First, implement the necessary business logic available via web interface to the researcher in order to manage all necessary parts of the calibration process. Second is to consolidate all the necessary data in order to configure the box for calibration.

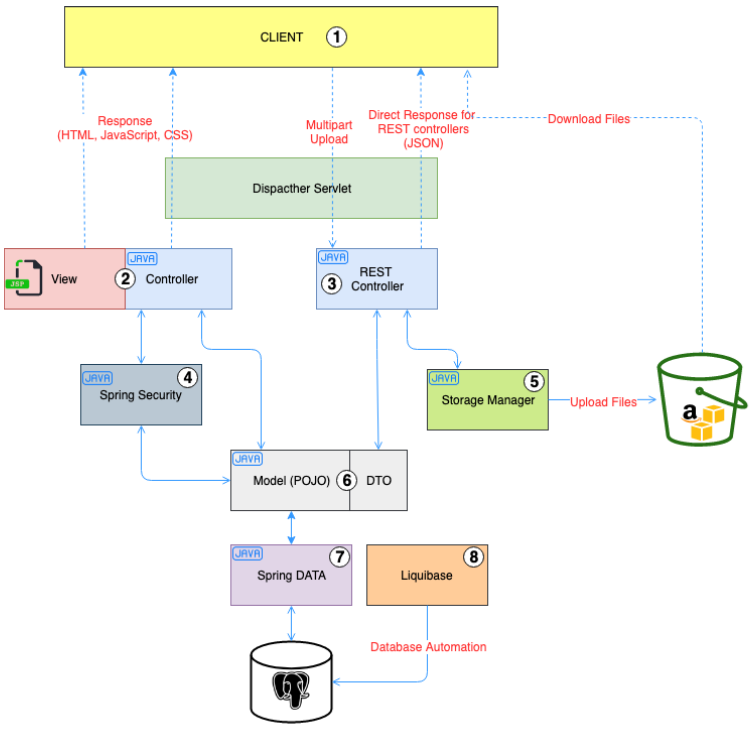


Figure 3.2: Web Application providing web management and web services persisting files on Amazon S3 and PostgreSQL

In order to achieve this purpose the chosen architecture (Figure 3.2) was based on a relational database supported by PostgreSQL. The database definition was made using Liquibase (Figure 3.2, 8). Liquibase was also used to bulk insert Biomark fact table data. The data is managed by Spring Data (Figure 3.2, 7), using the data model (Figure 3.2, 6). This data model respects the Liquibase database definition in the same way that Data Transfer Object(DTO) respects the client server defined protocol in order to allow the serialization/de-serialization during the REST(JSON) communication.

The user (Figure 3.2, 1) web interface is a server side generated set of pages using Spring MVC (Figure 3.2, 2) and Tiles2 for UI/UX Templating.

Regarding security, it was defined that the application should be able to allow two different roles (ADMIN, USER). To allow role/user management it was used another Spring Boot Framework called Spring Security (Figure 3.2, 4). This framework also uses the Model Plain Old Java Object's (POJOs) with the definition of the User and Role model (Java Classes).

By other side, the REST communication between the server side application and the CBs (Figure 3.2, 1) is ensured by another set of controllers (Figure 3.2, 3) using the DTOs and token based security. This REST controller's also implements the interface with Amazon S3 Cloud Storage (Figure 3.2, 5).

3.1.2 Calibration Box Software Architecture

The Calibration Box software implements the necessary logic to automate all the calibration process. To do this, the implementation should be able to control external connected devices, like the Crison pH Meter, and the Legato 100 pump, and also alternate between each membrane, collecting it's data and uploading for further analysis.

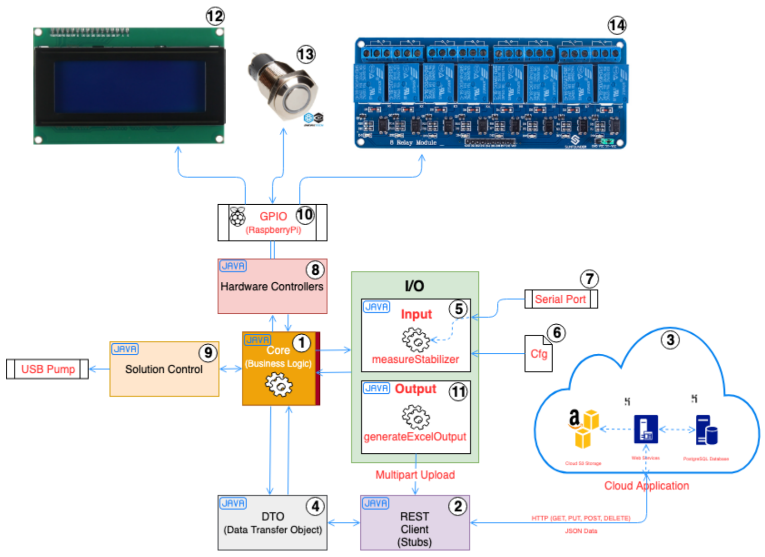


Figure 3.3: Calibration Box (CB) Software Architecture.

In the figure 3.3 representing CB software architecture, the main logic resides on the "Core" (figure 3.3, 1) module that interacts with the remaining modules in order to automate all the membrane calibration process. This main module requests the box status using its REST module (figure 3.3, 2) in a configurable interval. If the box status changes to "Assigned" the box requests its job. This Job comes in a JSON response that is serialized to its specific object using the shared DTO(figure 3.3, 4) between the CB and the server side software(figure 3.3, 3). After receiving the job, the CB changes its status using the REST module to "Working" and starts the interaction with the I/O module, first using the Input sub-module (figure 3.3, 5) in order to fetch from the configuration file the necessary operation configurations (figure 3.3, 6). To start measuring the membrane mV results, the CB interacts with the RaspberryPi GPIO interface (figure 3.3, 10) using the "Hardware Controller" module (figure 3.3, 8) to commute to the membrane using the relay board (figure 3.3, 14) disconnecting all the ports unless the necessary to measure using the serial interface (figure 3.3, 7). The input measure receives from the server side configuration the stability criteria and the number of retries in order to proceed to the next port in case of measuring an unstable membrane. After read a set of membranes, it is time to interact with the user to add an extra quantity of fluid to the existing solution (in the manual mode), or interact with the automatic pump (in the automatic mode) using the "Solution Control" module (figure 3.3, 9). When in the manual mode, the user will interact with the CB using LCD display (figure 3.3, 12) to receive instructions, and with the button to confirm the action taken (figure 3.3, 13). This button also has also a secondary function, that is to blink using its internal LED in order to advice the user that something needs a manual action. During the job execution, the CB keeps in memory all the collected data. Once finished, the CB generates locally an excel file with the collected data organized in the same format already used in the manual process (figure 3.3, 11). This excel file is then uploaded using the REST module via multi-part upload to the server. The job is then completed changing the box status to "FREE" again to allow the next job execution.

This workflow could be better understood with the following flowchart (figure 3.4).

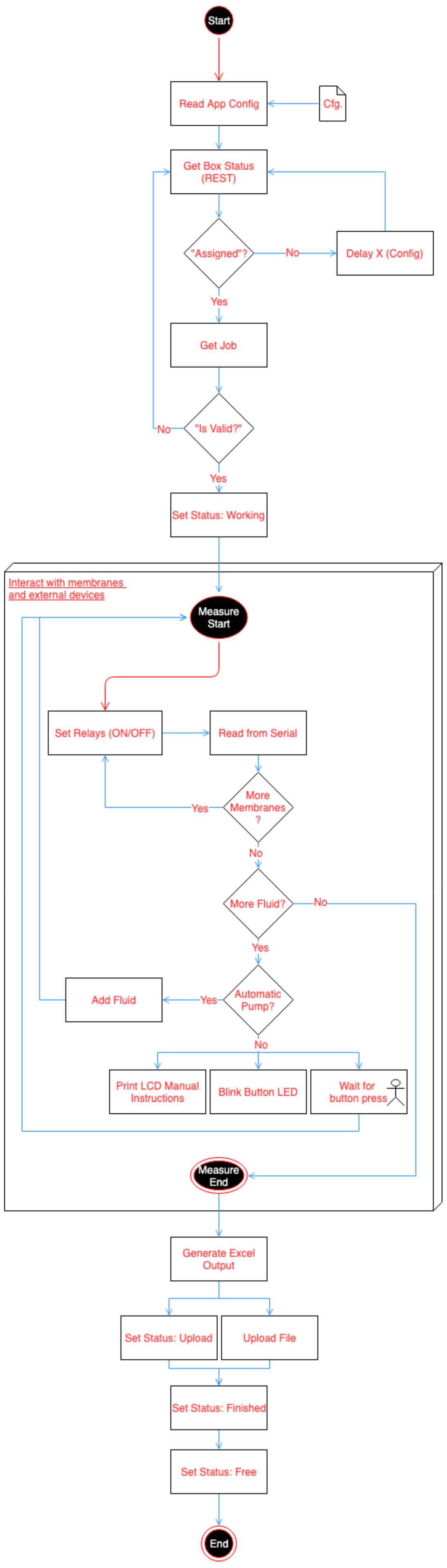


Figure 3.4: Calibration Box (CB) Flow Diagram.

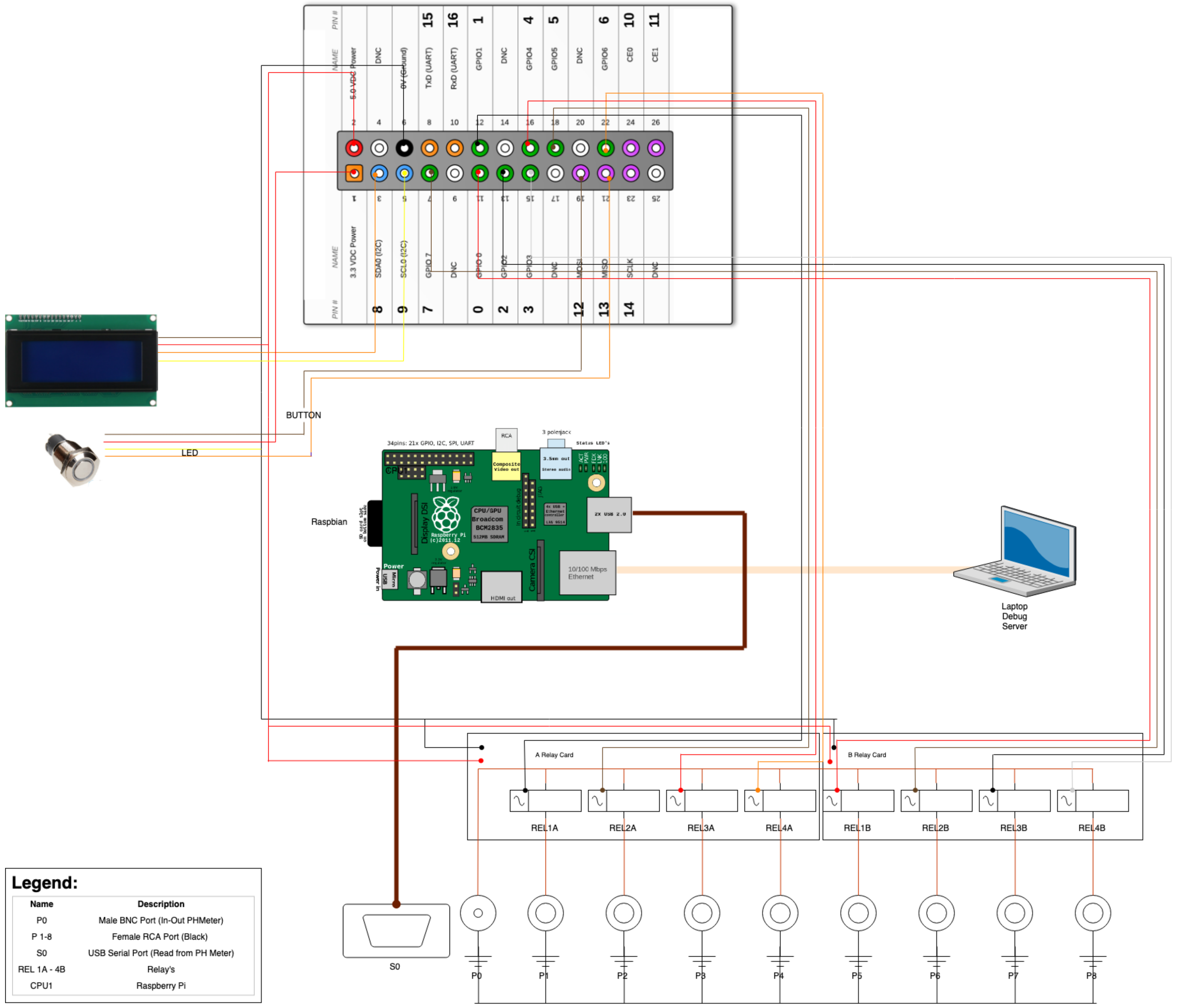
3.2 Hardware Architecture

The hardware architecture was based on the RaspberryPi platform using the GPIO to interact to the necessary hardware modules.

The following components was used in this hardware architecture and assembly:

- Raspberry Pi (Computer);
- SD Card;
- I2C 4x20 LCD Display;
- Touch button with LED;
- Two 4 Port relay boards;
- USB-Serial Interface;
- 8 RCA male ports (wall mounted);
- 1 BNC female port (cable mounted);
- Coaxial and direct cooper cables;
- Telecommunication (DSLAM) plastic box;
- Glue, Screws and other fixation solutions;

The schematic in figure 3.5 on page 38, represents a 8 port calibration box. The number of ports depend on the number of relays present/connected. In this case we have two relay boards with 4 relays each. All the relays are connected using the GPIO digital pins. The 4x20 display, is connected using the I2C Interface. The button uses two different digital pins. One to probe the button click action, and the other to blink the button LED light. More details about the connected hardware modules at the appendix A.



Legend:

Name	Description
P0	Male BNC Port (In-Out PHMeter)
P 1-8	Female RCA Port (Black)
S0	USB Serial Port (Read from PH Meter)
REL 1A - 4B	Relay's
CPU1	Raspberry Pi

Figure 3.5: Calibration Box (CB) Hardware Diagram.

Chapter 4

Server Application

4.1 Server

The server side application allow the researcher and the provisioning team to insert the necessary materials and configurations in order to configure the calibration process and submit them to a Calibration Box. There is also an REST API developed to provide the necessary integration with the CB's.

In this section the server side development process is described as long the technologies used and some sample examples of code necessary to explain important algorithms or approaches.

4.1.1 Development process and frameworks

This project was developed using OpenJDK 8 [16].

Since the server-side application is a Java based application, the most mature and stable solution to manage the project structure and dependencies is Maven [17]. The Maven main role in this project was to manage the dependencies and also the modules organization

The following dependencies was used to develop the server side web application:

Dependency	Version	License
commons-math3	3.5	Apache License, Version 2.0 [18]
spring-boot-starter-data-jpa	2.1.7.RELEASE	Apache License, Version 2.0 [18]
spring-boot-starter-web	2.1.7.RELEASE	Apache License, Version 2.0 [18]
spring-boot-starter-security	2.1.7.RELEASE	Apache License, Version 2.0 [18]
spring-boot-starter-hateoas	2.1.7.RELEASE	Apache License, Version 2.0 [18]
spring-boot-starter-test	2.1.7.RELEASE	Apache License, Version 2.0 [18]
springfox-swagger2	2.9.2	Apache License, Version 2.0 [18]
springfox-core	2.9.2	Apache License, Version 2.0 [18]
springfox-swagger-ui	2.9.2	Apache License, Version 2.0 [18]
tomcat-embed-core	9.0.14	Apache License, Version 2.0 [18]
tomcat-embed-jasper	9.0.14	Apache License, Version 2.0 [18]
tomcat-jasper	9.0.14	Apache License, Version 2.0 [18]
tomcat-jasper-el	9.0.14	Apache License, Version 2.0 [18]
tomcat-jsp-api	9.0.14	Apache License, Version 2.0 [18]
tiles-jsp	3.0.5	Apache License, Version 2.0 [18]
liquibase-core	3.6.2	Apache License, Version 2.0 [18]
aws-java-sdk	1.11.133	Apache License, Version 2.0 [18]
spring-boot-starter	2.1.7.RELEASE	Apache License, Version 2.0 [18]
modelmapper	2.3.2	Apache License, Version 2.0 [18]
hibernate-validator	6.0.13.Final	Apache License, Version 2.0 [18]
javamelody-spring-boot-starter	1.75.0	Apache License, Version 2.0 [18]
logback-classic	1.1.3	Eclipse Public License - v 1.0 [19]
jstl	1.2	CDDL Version 1.1 [20]
postgresql	42.2.1	BSD-2-Clause[21]
jcl-over-slf4j	1.7.12	MIT License [22]
lombok	1.18.4	The MIT License [22]

Table 4.1: Server Side Dependencies and Licenses

Regarding tools, this project was developed using Docker (to provide the local simulation environment), pgAdmin (to manage the database development), JetBrains IntelliJ IDEA Integrated Development Environment (IDE)(licensed by the free education license due to the *Instituto Superior de Engenharia do Porto (ISEP)* partnership).

For many times it was necessary to change the entire stack data model, webpages and other, ending on a new deployment to the cloud provider in order to allow the necessary remote testing. Due to this, it was implemented a CI/CD solution. Since this project development version control was from the beginning made using Git at the public free Gitlab.com service, it was the best approach to prepare a pipeline to automatically deliver the new update via the public cloud service every time an update was merged onto the main development line using the GitLab CI/CD functionality.

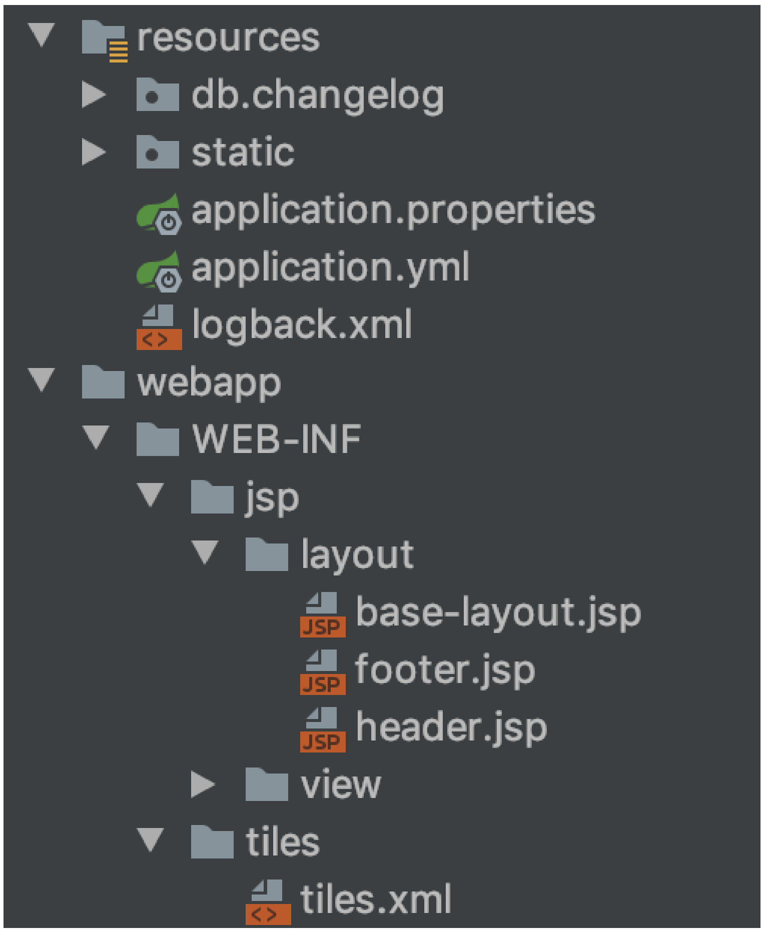
The GitLab CI configuration:

```
stages:
- DEPLOY

Deploy to Heroku:
stage: DEPLOY
tags:
- all
script:
- heroku config:set MAVEN_CUSTOM_GOALS="clean package
   spring-boot:repackage" -a biomark
- heroku git:remote -a biomark
- heroku plugins:install heroku-repo
- heroku repo:purge_cache -a biomark
- git commit --allow-empty -m "Purge cache"
- git push heroku HEAD:master --force
```

The heroku serverless deployment automatically detects the Java application and sets it's profile, but the spring dependencies has their own way to compile, using it's own plugin, and a goal to ensure the final JAR will contain the necessary conditions to run. The 'MAVEN_CUSTOM_GOALS' configuration var is used by the Heroku cloud to run the defined command instead the autodetect for generic Java Applications. Heroku also have it's own Git repository and the pipeline automatically pushes the code to the remote git. Since some times Heroku keeps serving outdated application Jar file for a while, there is a 'purge_cache' to ensure that the new deployment is used and the old deployment should be automatically removed (this requires some seconds of transitory downtime).

4.1.2 Web Front-End

The main page was developed using Spring MVC [23]. One of the main aspects that is important regarding a web page, is the usability and look. Due to this, and to be able to apply a web template, the chosen technology for templating was Apache Tiles [24]. For the Hypertext Markup Language (HTML), Cascading Style Sheets (CSS) and JavaScript (JS) administration layout page there was used a free template downloaded from Creative-Tim called "paper-dashboard" [25] and licensed with the MIT License [22]. With Apache Tiles, all the layout settings stays centralized in the project folder -> resources -> webapp -> WEB-INF -> jsp -> layout (figure: 4.1).



```
▼ resources
    ▶ db.changelog
    ▶ static
      application.properties
      application.yml
      logback.xml
▼ webapp
    ▼ WEB-INF
        ▼ jsp
            ▼ layout
                  base-layout.jsp
                  footer.jsp
                  header.jsp
            ▶ view
        ▼ tiles
              tiles.xml
```

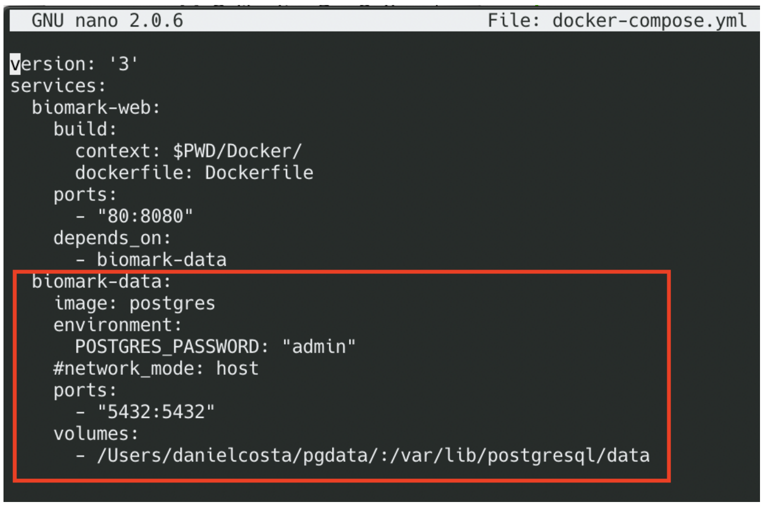
Figure 4.1: Server Frontend template and Tiles integration.

4.1.3 Database Model and Development

This was the one of the most time consuming parts of this project. The main reason for this is that it is impossible to design any data model without a clear view of all the project. Due to this, the data model design was the trigger for a lot of meetings and adjustments in the process in order to better understand the relationship between all the tables.

The Biomark process is a well defined operational process, and the chosen technology to store all the data was a relational database. PostgreSQL [26] is open-source software released with it's own license [27], Cloud friendly, stable, mature and with a good performance even in a small environment. Another reason to choose PostgreSQL is their support for JSON fields, not used in the scope of this thesis, but something useful for the data analysis development (Future work).

For the initial development, there was locally executed a Docker Container [28] with the official image from PostgreSQL (figure: 4.2), this deploy also allow the execution of the server side deployment in order to test locally all the server side environment.



```
  GNU nano 2.0.6                         File: docker-compose.yml

version: '3'
services:
  biomark-web:
    build:
      context: $PWD/Docker/
      dockerfile: Dockerfile
    ports:
      - "80:8080"
    depends_on:
      - biomark-data
  biomark-data:
    image: postgres
    environment:
      POSTGRES_PASSWORD: "admin"
    #network_mode: host
    ports:
      - "5432:5432"
    volumes:
      - /Users/danielcosta/pgdata/:/var/lib/postgresql/data
```

Figure 4.2: PostgreSQL local Docker based deployment.

To run this container we need to execute the following command (having docker and docker-compose installed).

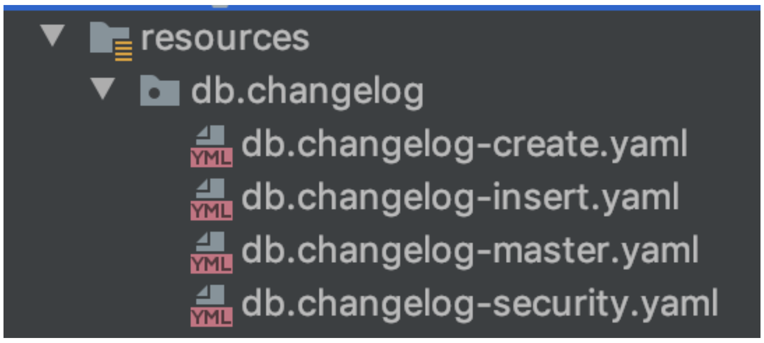
```
docker-compose up biomark-data
```

To create the base database structure and insert the initial data in the fact tables, the chosen technology was Liquibase. Liquibase is a data automation solution that takes care about the database changesets using their own tables to manage the database structure and changes.

Fist, there are two manual steps that should be done:

- Create the "biomark" database;
- Create the "biomark" schema inside the "biomark" database;

The next step is to run the server application, that, during the boot applies all the liquibase change sets present o the folder resources -> db -> changelog (figure: 4.3.



```
▼ resources
  ▼ db.changelog
      db.changelog-create.yaml
      db.changelog-insert.yaml
      db.changelog-master.yaml
      db.changelog-security.yaml
```

Figure 4.3: Project database change log files.

The database change log files, were organized in the following structure:

- "db.changelog-master.yaml" - This is the main file that calls the remaining above files;
- "db.changelog-create.yaml" - This file has all the table creation and relations statements;
- "db.changelog-insert.yaml" - This file has all the data insertion to the fact tables logic;
- "db.changelog-security.yaml" - This file creates all the necessary initial security roles and users;

After the database/schema creation and during the server side application startup, the tables, relationships, indexes are also created, the fact tables and security settings are filled. The "biomark" schema contains all the application logic tables. The schema "public" contain all the liquibase tables to keep the changeset trackable and up to date.

It was time to generate a database diagram. To do this, the application used was the "schema crawler" running inside a docker container. To execute this docker container the following command was executed:

```
docker run \
-v $(pwd):/share \
-it \
--network host \
--name schemacrawler \
--entrypoint=/bin/bash \
schemacrawler/schemacrawler
```

And after the schema crawler container starts:

```
schemacrawler --info-level=standard --server=postgresql
   --host=localhost --port=5432 --database=biomark
   --schemas=biomark --user=postgres --password=admin
   --infolevel=detailed --command=schema --outputformat=png
   --outputfile=/share/dbDiagram.png
```

The output of this command is the diagram present on figure 4.4.

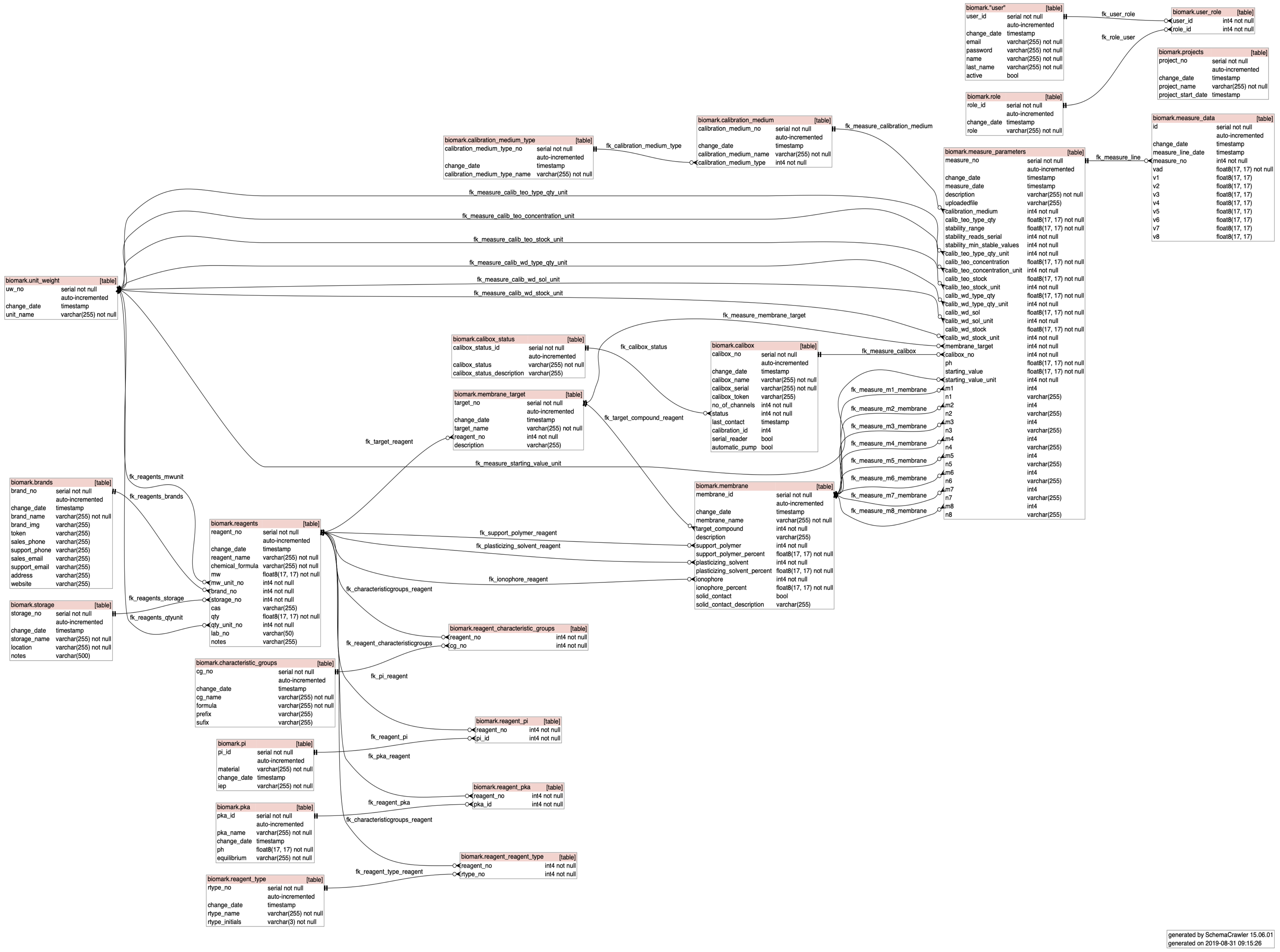
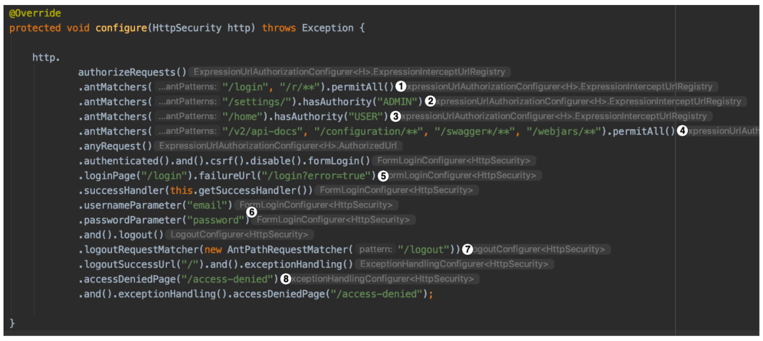

biomark."user" [table]
user_id serial not null auto-incremented
change_date timestamp
email varchar(255) not null
password varchar(255) not null
name varchar(255) not null
last_name varchar(255) not null
active bool
fk_user_role
fk_role_user
biomark.user_role [table]
user_id int4 not null
role_id int4 not null
biomark.projects [table]
project_no serial not null auto-incremented
change_date timestamp
project_name varchar(255) not null
project_start_date timestamp
biomark.role [table]
role_id serial not null auto-incremented
change_date timestamp
role varchar(255) not null
biomark.measure_data [table]
id serial not null auto-incremented
change_date timestamp
measure_line_date timestamp
measure_no int4 not null
vad float8(17, 17) not null
v1 float8(17, 17)
v2 float8(17, 17)
v3 float8(17, 17)
v4 float8(17, 17)
v5 float8(17, 17)
v6 float8(17, 17)
v7 float8(17, 17)
v8 float8(17, 17)
fk_measure_line
fk_measure_calibration_medium
biomark.calibration_medium [table]
calibration_medium_no serial not null auto-incremented
change_date timestamp
calibration_medium_name varchar(255) not null
calibration_medium_type int4 not null
fk_calibration_medium_type
biomark.calibration_medium_type [table]
calibration_medium_type_no serial not null auto-incremented
change_date timestamp
calibration_medium_type_name varchar(255) not null
biomark.measure_parameters [table]
measure_no serial not null auto-incremented
change_date timestamp
measure_date timestamp
description varchar(255) not null
uploadedfile varchar(255)
calibration_medium int4 not null
calib_teo_type_qty float8(17, 17) not null
stability_range float8(17, 17) not null
stability_reads_serial int4 not null
stability_min_stable_values int4 not null
calib_teo_type_qty_unit int4 not null
calib_teo_concentration float8(17, 17) not null
calib_teo_concentration_unit int4 not null
calib_teo_stock float8(17, 17) not null
calib_teo_stock_unit int4 not null
calib_wd_type_qty float8(17, 17) not null
calib_wd_type_qty_unit int4 not null
calib_wd_sol float8(17, 17) not null
calib_wd_sol_unit int4 not null
calib_wd_stock float8(17, 17) not null
calib_wd_stock_unit int4 not null
membrane_target int4 not null
calibox_no int4 not null
ph float8(17, 17) not null
starting_value float8(17, 17) not null
starting_value_unit int4 not null
m1 int4
n1 varchar(255)
m2 int4
n2 varchar(255)
m3 int4
n3 varchar(255)
m4 int4
n4 varchar(255)
m5 int4
n5 varchar(255)
m6 int4
n6 varchar(255)
m7 int4
n7 varchar(255)
m8 int4
n8 varchar(255)
fk_measure_calib_teo_type_qty_unit
fk_measure_calib_teo_concentration_unit
fk_measure_calib_teo_stock_unit
fk_measure_calib_wd_type_qty_unit
fk_measure_calib_wd_sol_unit
fk_measure_calib_wd_stock_unit
fk_measure_membrane_target
fk_measure_starting_value_unit
biomark.unit_weight [table]
uw_no serial not null auto-incremented
change_date timestamp
unit_name varchar(255) not null
biomark.calibox_status [table]
calibox_status_id serial not null auto-incremented
calibox_status varchar(255) not null
calibox_status_description varchar(255)
fk_calibox_status
biomark.calibox [table]
calibox_no serial not null auto-incremented
change_date timestamp
calibox_name varchar(255) not null
calibox_serial varchar(255) not null
calibox_token varchar(255)
no_of_channels int4 not null
status int4 not null
last_contact timestamp
calibration_id int4
serial_reader bool
automatic_pump bool
fk_measure_calibox
fk_measure_m1_membrane
fk_measure_m2_membrane
fk_measure_m3_membrane
fk_measure_m4_membrane
fk_measure_m5_membrane
fk_measure_m6_membrane
fk_measure_m7_membrane
fk_measure_m8_membrane
biomark.membrane_target [table]
target_no serial not null auto-incremented
change_date timestamp
target_name varchar(255) not null
reagent_no int4 not null
description varchar(255)
fk_target_reagent
fk_target_compound_reagent
biomark.membrane [table]
membrane_id serial not null auto-incremented
change_date timestamp
membrane_name varchar(255) not null
target_compound int4 not null
description varchar(255)
support_polymer int4 not null
support_polymer_percent float8(17, 17) not null
plasticizing_solvent int4 not null
plasticizing_solvent_percent float8(17, 17) not null
ionophore int4 not null
ionophore_percent float8(17, 17) not null
solid_contact bool
solid_contact_description varchar(255)
fk_support_polymer_reagent
fk_plasticizing_solvent_reagent
fk_ionophore_reagent
fk_reagents_mwunit
fk_reagents_brands
biomark.brands [table]
brand_no serial not null auto-incremented
change_date timestamp
brand_name varchar(255) not null
brand_img varchar(255)
token varchar(255)
sales_phone varchar(255)
support_phone varchar(255)
sales_email varchar(255)
support_email varchar(255)
address varchar(255)
website varchar(255)
biomark.reagents [table]
reagent_no serial not null auto-incremented
change_date timestamp
reagent_name varchar(255) not null
chemical_formula varchar(255) not null
mw float8(17, 17) not null
mw_unit_no int4 not null
brand_no int4 not null
storage_no int4 not null
cas varchar(255)
qty float8(17, 17) not null
qty_unit_no int4 not null
lab_no varchar(50)
notes varchar(255)
fk_reagents_storage
fk_reagents_qtyunit
biomark.storage [table]
storage_no serial not null auto-incremented
change_date timestamp
storage_name varchar(255) not null
location varchar(255) not null
notes varchar(500)
fk_characteristicgroups_reagent
biomark.reagent_characteristic_groups [table]
reagent_no int4 not null
cg_no int4 not null
fk_reagent_characteristicgroups
biomark.characteristic_groups [table]
cg_no serial not null auto-incremented
change_date timestamp
cg_name varchar(255) not null
formula varchar(255) not null
prefix varchar(255)
sufix varchar(255)
fk_pi_reagent
biomark.reagent_pi [table]
reagent_no int4 not null
pi_id int4 not null
fk_reagent_pi
biomark.pi [table]
pi_id serial not null auto-incremented
material varchar(255) not null
change_date timestamp
iep varchar(255) not null
fk_pka_reagent
biomark.reagent_pka [table]
reagent_no int4 not null
pka_id int4 not null
fk_reagent_pka
biomark.pka [table]
pka_id serial not null auto-incremented
pka_name varchar(255) not null
change_date timestamp
ph float8(17, 17) not null
equilibrium varchar(255) not null
fk_characteristicgroups_reagent
biomark.reagent_reagent_type [table]
reagent_no int4 not null
rtype_no int4 not null
fk_reagent_type_reagent
biomark.reagent_type [table]
rtype_no serial not null auto-incremented
change_date timestamp
rtype_name varchar(255) not null
rtype_initials varchar(3) not null
generated by SchemaCrawler 15.06.01
generated on 2019-08-31 09:15:26

Figure 4.4: Database diagram.

4.1.4 Security

Regarding security, the main requirements are not complex due to an open mindset present on the BioMark laboratory. Beside this, there is a need to protect some software administrative functionalities, like create users and manage runtime settings that should be done by an administrator and not by every user. To achieve this goal, and since the whole solution was based on the Spring Framework, the security solution was based on Spring Security [29].

Spring security integrates with the remaining Spring frameworks like MVC, allowing the appropriate role based security. To configure, the Spring Security requires some detailed specification about what is necessary for the login, what to do if the login fails, and what resources need a specific role or authentication. The server security configuration can be visualized on the explained figure 4.5.



```
@Override
protected void configure(HttpSecurity http) throws Exception {

    http.
            authorizeRequests() ExpressionUrlAuthorizationConfigurer<H>.ExpressionInterceptUrlRegistry
            .antMatchers( ...antPatterns: "/login", "/r/**").permitAll() 1 xpressionUrlAuthorizationConfigurer<H>.ExpressionInterceptUrlRegistry
            .antMatchers( ...antPatterns: "/settings/").hasAuthority("ADMIN") 2 xpressionUrlAuthorizationConfigurer<H>.ExpressionInterceptUrlRegistry
            .antMatchers( ...antPatterns: "/home").hasAuthority("USER") 3 xpressionUrlAuthorizationConfigurer<H>.ExpressionInterceptUrlRegistry
            .antMatchers( ...antPatterns: "/v2/api-docs", "/configuration/**", "/swagger*/**", "/webjars/**").permitAll() 4 xpressionUrlAuth
            .anyRequest() ExpressionUrlAuthorizationConfigurer<H>.AuthorizedUrl
            .authenticated().and().csrf().disable().formLogin() FormLoginConfigurer<HttpSecurity>
            .loginPage("/login").failureUrl("/login?error=true") 5 ormLoginConfigurer<HttpSecurity>
            .successHandler(this.getSuccessHandler()) FormLoginConfigurer<HttpSecurity>
            .usernameParameter("email") FormLoginConfigurer<HttpSecurity>
            .passwordParameter("password") 6 FormLoginConfigurer<HttpSecurity>
            .and().logout() LogoutConfigurer<HttpSecurity>
            .logoutRequestMatcher(new AntPathRequestMatcher( pattern: "/logout")) 7 ogoutConfigurer<HttpSecurity>
            .logoutSuccessUrl("/").and().exceptionHandling() ExceptionHandlingConfigurer<HttpSecurity>
            .accessDeniedPage("/access-denied") 8 xceptionHandlingConfigurer<HttpSecurity>
            .and().exceptionHandling().accessDeniedPage("/access-denied");

}
```

Figure 4.5: Spring Security configuration

The figure 4.5 represents the server side security configuration, where the line 1 and 4 contains the endpoints who do not need any kind of authentication to be accessed. The /login contains the login page where the user is able to authenticate on the application. The /r/** represents the REST services that are token based authenticated. The remaining endpoints on the line 4 are related with the online API documentation and monitoring endpoints. The line 2 and 3 represents the role specific pages, those pages requires their specific role on the user. Any other request requires user authentication but do not depend on the role specific permission.The line number 5 declares the login page and the login failure URL in order to notice the user for the wrong credentials inserted on the fields declared with the number 6. The line with number 7 is the logout page that expires the session based login. The line 8 represents the page that will be displayed to the user who tries to access any page without authentication.

To provide the database based authentication, there are two database queries associated with the authentication.

The user query, used for the user authentication and validation:

```
select email, password, active from biomark.user where email=?
```

And, the Roles per User query:

```
select u.email, r.role from biomark.user u inner join
    biomark.user_role ur on(u.user_id=ur.user_id) inner join
    biomark.role r on(ur.role_id=r.role_id) where u.email=?
```

Both of these queries are configurable on the "application.properties" file.

The login page (figure: 4.6) is based on the Bootstrap [30] web framework backed by its own Spring MVC controller.

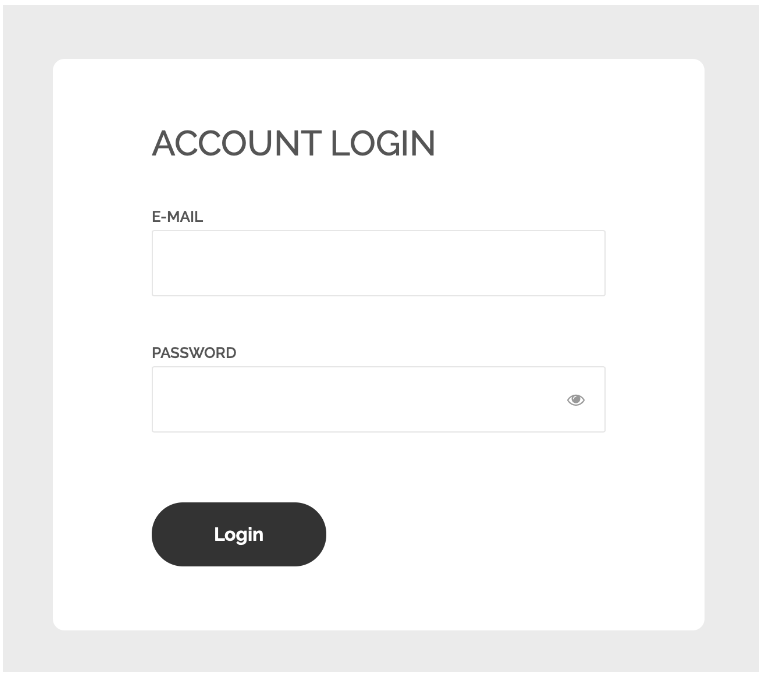


Figure 4.6: BioMark login page

4.1.5 Monitoring and Performance

With the Cloud deployment in mind, the server requirements and the workload used to achieve the main processes are a concern since there could be extended costs with the platform if there is a memory leak or any other hardware resources consumption concern. To provide enough monitoring indicators that Java already collects about the Java Virtual Machine (JVM) running applications and making them available to the users with the "ADMIN" role, the JavaMelody [31] was the chosen framework.

JavaMelody provides a simple integration with the application, with low footprint since it only displays the indicators already collected by the Java Virtual Machine.

In the user interface header there is a link to the JavaMelody monitoring page (figure: 4.7).

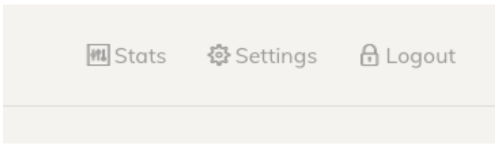


Figure 4.7: Monitoring link at the webpage header

The admin users are allowed to monitor the server performance with a clear graphical interface with all the necessary indicators, like, CPU, Memory, Garbage Collection, Database Connections and errors (figure: 4.8).

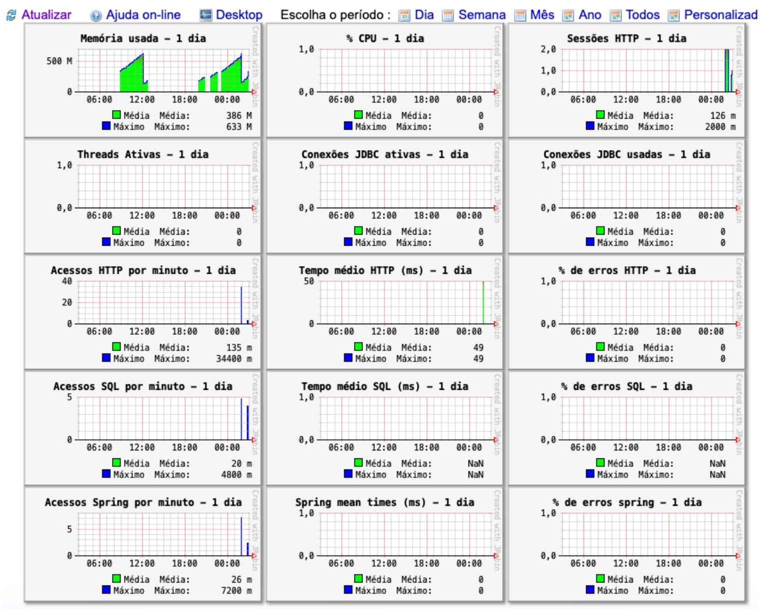


Figure 4.8: JavaMelody monitoring collected data

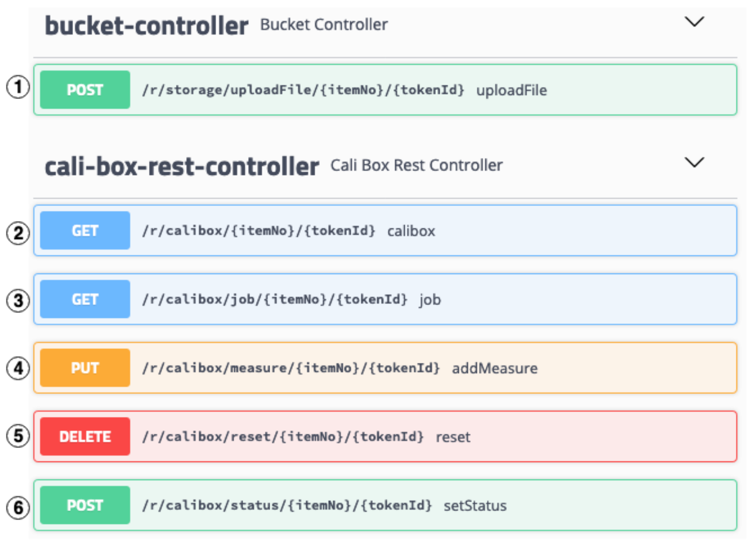
This monitoring solution allowed the correct sizing of the server less instance using the smallest instance with 512Mb memory.

4.1.6 REST API Services

To provide the necessary backend for the Calibration Box (CB) there was developed a REST API with all the necessary support solutions.

This REST API was developed integrated with all the remaining server side solution using Spring Boot. There was also implemented the API documentation and test interface using Swagger [32].

To organize the REST endpoints 4.9 the root "/r/" was created with all the REST API endpoints and divided in two sub-roots, one for the Calibration Box called "/calibox/" and another for the storage results file uploads "/storage/".



bucket-controller Bucket Controller

① POST `/r/storage/uploadFile/{itemNo}/{tokenId}` uploadFile

cali-box-rest-controller Cali Box Rest Controller

② GET `/r/calibox/{itemNo}/{tokenId}` calibox

③ GET `/r/calibox/job/{itemNo}/{tokenId}` job

④ PUT `/r/calibox/measure/{itemNo}/{tokenId}` addMeasure

⑤ DELETE `/r/calibox/reset/{itemNo}/{tokenId}` reset

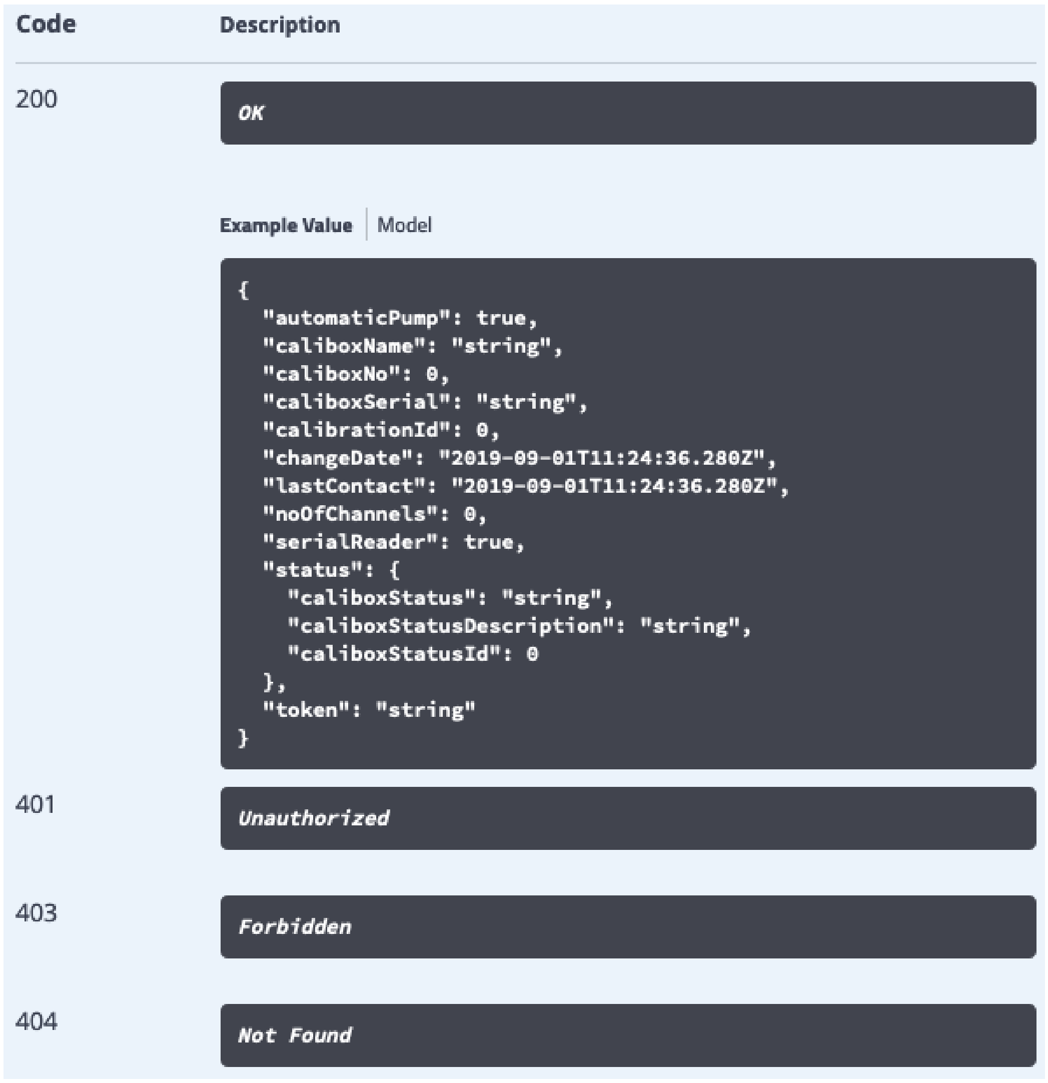
⑥ POST `/r/calibox/status/{itemNo}/{tokenId}` setStatus

Figure 4.9: REST Backend Endpoints

In all of those endpoints present on the figure 4.9 the parameters "{itemNo}" represents the CB unique id, and the "{tokenId}" represents its authentication token.

The figure 4.9 line 1 represents the storage endpoint used to upload using multipart file upload the resultant file of a calibration. This endpoints requires the two above parameters and a multipart upload

The line 2 of the same image represents the endpoint most called from a working Calibration Box. This endpoint is called every 5 seconds (configurable in the CB) to get the CB status (see figure: 4.10).



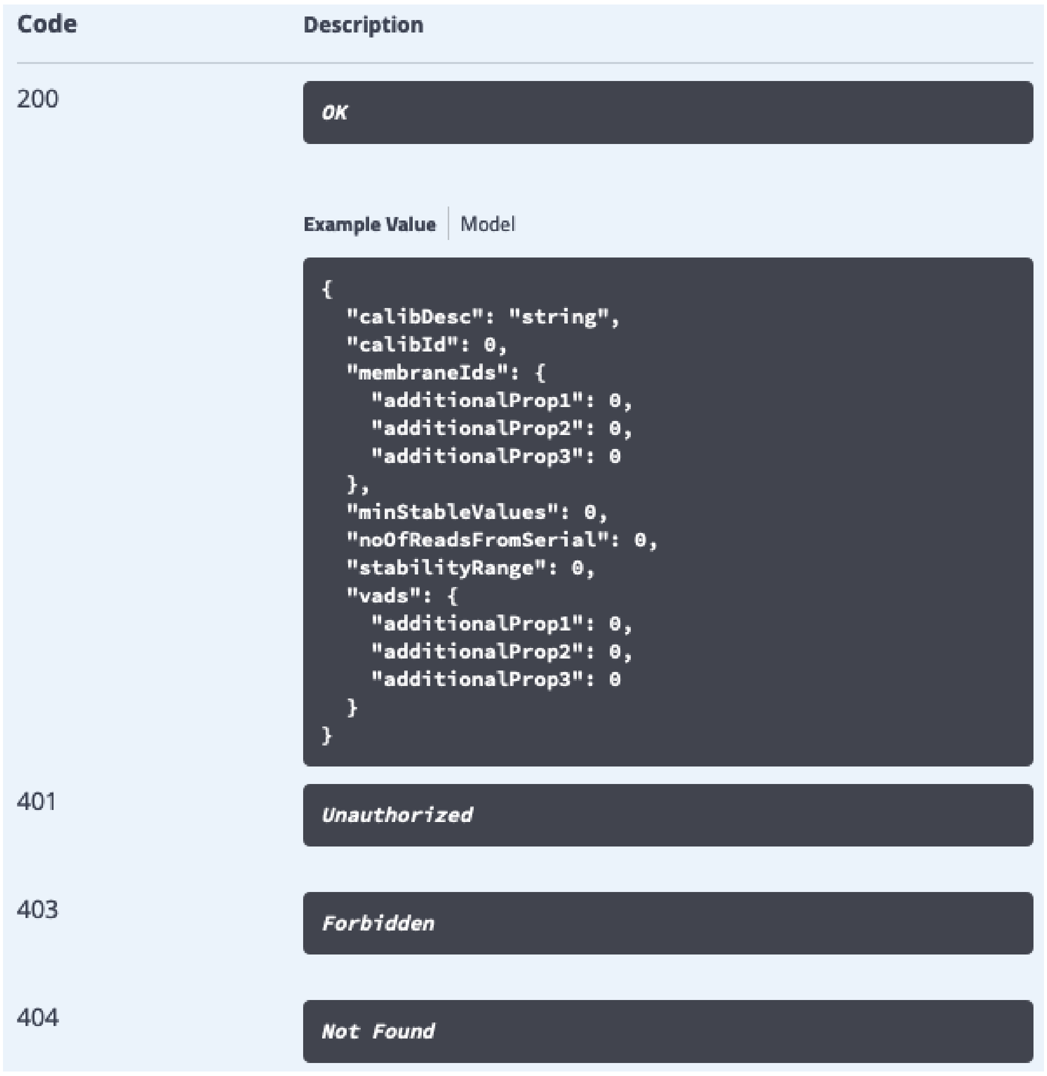
Code	Description
200	*OK*
401	*Unauthorized*
403	*Forbidden*
404	*Not Found*

Example Value | Model

```
{
  "automaticPump": true,
  "caliboxName": "string",
  "caliboxNo": 0,
  "caliboxSerial": "string",
  "calibrationId": 0,
  "changeDate": "2019-09-01T11:24:36.280Z",
  "lastContact": "2019-09-01T11:24:36.280Z",
  "noOfChannels": 0,
  "serialReader": true,
  "status": {
    "caliboxStatus": "string",
    "caliboxStatusDescription": "string",
    "caliboxStatusId": 0
  },
  "token": "string"
}
```

Figure 4.10: CB REST Status Response

After the CB status was changed (due to an assigned job) the CB requests from the endpoint (figure 4.9 line 3) the job to be done. This endpoint returns the minimum necessary data (see figure 4.11).


Code | Description

200 | OK

Example Value | Model

```
{
  "calibDesc": "string",
  "calibId": 0,
  "membraneIds": {
    "additionalProp1": 0,
    "additionalProp2": 0,
    "additionalProp3": 0
  },
  "minStableValues": 0,
  "noOfReadsFromSerial": 0,
  "stabilityRange": 0,
  "vads": {
    "additionalProp1": 0,
    "additionalProp2": 0,
    "additionalProp3": 0
  }
}
```

401 | Unauthorized

403 | Forbidden

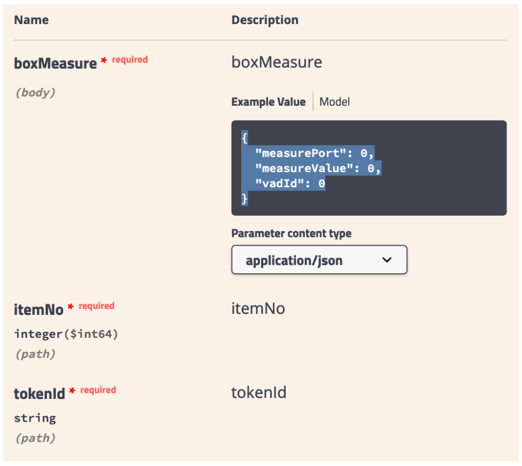
404 | Not Found

Figure 4.11: CB REST Job Response

The membrane Id's is an array with String, Integer pair containing the membrane name and its unique Id. The Vad's is an array with all the additional values of volume to be added automatically or by the user during the calibration process. The "minStableValues", "noOfReadsFromSerial", "stabilityRange" are parameters configurable by the user to define the stability criteria.

During the calibration process, the CB is able to send each measured data to the server endpoint (figure 4.9 line 4) . By default this endpoint is not used, since to reduce cloud usage and costs, the CB only sends all the collected data at the end via file upload.

This endpoint accepts the following data (figure 4.12, A), after submit the data the REST the server endpoint responds with the following (figure 4.12, B).



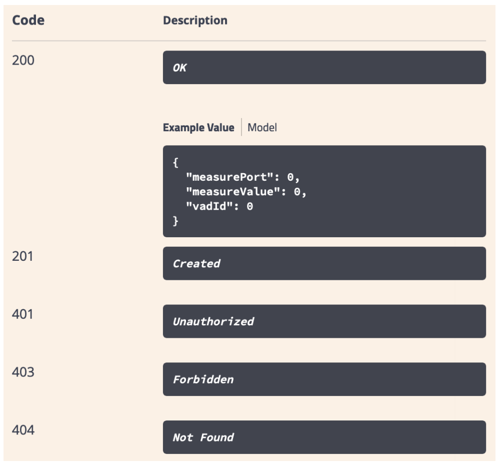
Name	Description
boxMeasure * required (body)	boxMeasure
itemNo * required integer($int64) (path)	itemNo
tokenId * required string (path)	tokenId

Example Value | Model

```
{
  "measurePort": 0,
  "measureValue": 0,
  "vadId": 0
}
```

Parameter content type: application/json

(a) Request



Code	Description
200	OK
201	Created
401	Unauthorized
403	Forbidden
404	Not Found

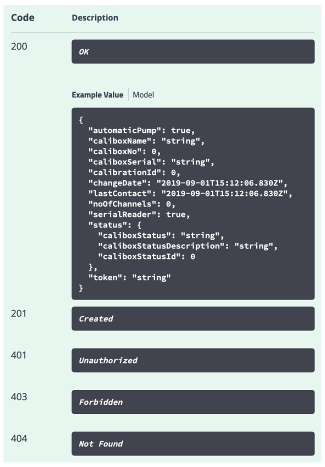
Example Value | Model

```
{
  "measurePort": 0,
  "measureValue": 0,
  "vadId": 0
}
```

(b) Response

Figure 4.12: Add Mesure REST.

Every time CB waits for user interaction, finishes their job, or other status change, the CB updates (figure: 4.13) it's status using a REST endpoint (figure 4.9 line 6).

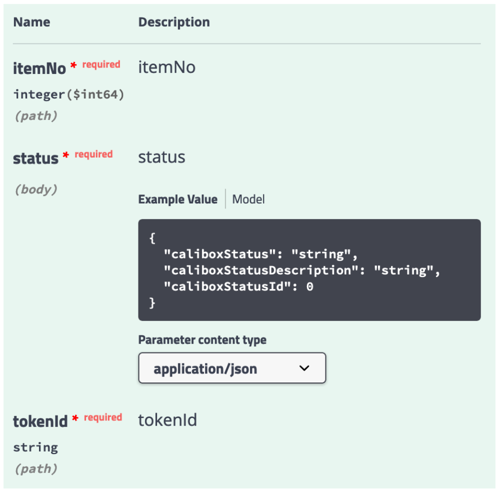


Code	Description
200	OK
201	Created
401	Unauthorized
403	Forbidden
404	Not Found

Example Value | Model

```
{
  "automaticPump": true,
  "caliboxName": "string",
  "caliboxNo": 0,
  "caliboxSerial": "string",
  "calibrationId": 0,
  "changeDate": "2019-09-01T15:12:06.830Z",
  "lastContact": "2019-09-01T15:12:06.830Z",
  "noOfChannels": 0,
  "serialReader": true,
  "status": {
    "caliboxStatus": "string",
    "caliboxStatusDescription": "string",
    "caliboxStatusId": 0
  },
  "token": "string"
}
```

(a) Request



Name	Description
itemNo * required integer($int64) (path)	itemNo
status * required (body)	status
tokenId * required string (path)	tokenId

Example Value | Model

```
{
  "caliboxStatus": "string",
  "caliboxStatusDescription": "string",
  "caliboxStatusId": 0
}
```

Parameter content type: application/json

(b) Response

Figure 4.13: Change CB status via REST.

By the end, CB finishes his job submitting the data, and resets its status back to the "free" state by calling the DELETE endpoint (figure: 4.9 line 5).

To serialize and de-serialize the data on client (CB) and the server (REST), both shared the same Data Transfer Object's with the following data model/POJO's:

```
BoxJob{
    calibDesc   string
    calibId integer($int64)
    membraneIds {
        < * >:  integer($int64)
    }
    minStableValues integer($int32)
    noOfReadsFromSerial integer($int32)
    stabilityRange  number($double)
    vads    {
        < * >:  number($double)
    }
}
```

```
CaliBoxStatus{
    caliboxStatus   string
    caliboxStatusDescription    string
    caliboxStatusId integer($int64)
}
```

```
CaliBox{
    automaticPump   boolean
    caliboxName string
    caliboxNo   integer($int64)
    caliboxSerial   string
    calibrationId   integer($int64)
    changeDate  string($date-time)
    lastContact string($date-time)
    noOfChannels    integer($int32)
    serialReader    boolean
    status  CaliBoxStatus{...}
    token   string
}
```

```
BoxMeasure{
    measurePort integer($int32)
    measureValue    number($double)
    vadId   integer($int64)
}
```

4.1.7 Storage in a Server Less Cloud Deployment

Currently the calibration results are prepared in a excel file automatically during the end of the calibration at the Calibration Box side for the following reasons:

- The on premisses costs are less then the cloud processing costs;
- The cloud storage costs are low for this kind of files and sizes;
- The file sizes are small;
- The researches are familiar with this format to process and analyse their data;
- An advanced data processing solution requires the double of this project time to implement;

But, in a server less cloud deployment architecture, all the data persistence should be done using dedicated services. For the database persistence the solution was the PostgreSQL instance. For the file store, the solution was the Amazon S3 storage bucket.

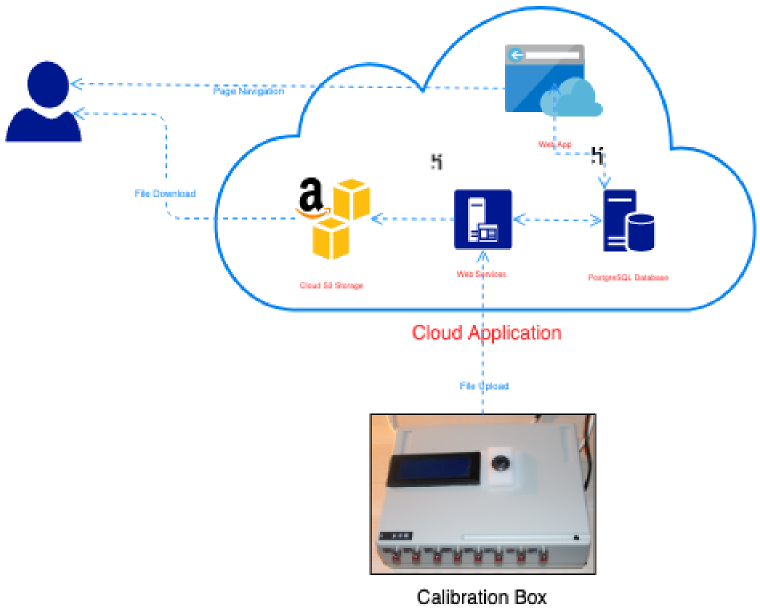


Figure 4.14: File Cycle from and to S3 Storage.

In the figure 4.14 is there is a representation of the file cycle. First it is produced by the Calibration process in the CB. The CB job execution thread uploads the file via web app REST interface. The server side application uploads the file to the Amazon S3 Storage getting it's file path and id, and saving it to the PostgreSQL database. Finally, when the researcher access web interface to retrieve the results, the download link is available downloading directly from the Amazon S3 public file address.

4.2 User Interface

The first and main goal of the user interface in this project is to allow user configure the calibration process in order to automate it.

To do this the user first login in the user interface with their account 4.6.

After login the web page is divided in three main parts (figure 4.15):

- Top or header with administrative access to the monitoring and settings (A);
- Left side menu with operational options (B1) and fact Create, Read, Update, Delete (CRUD) (B2);
- Content part of the page where lives all the application forms and listings (C);

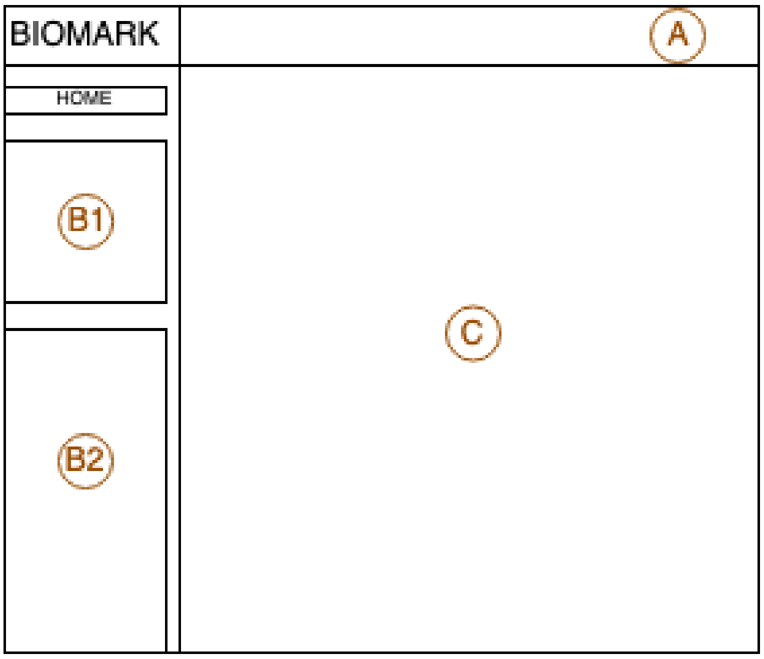


Figure 4.15: User Interface Layout.

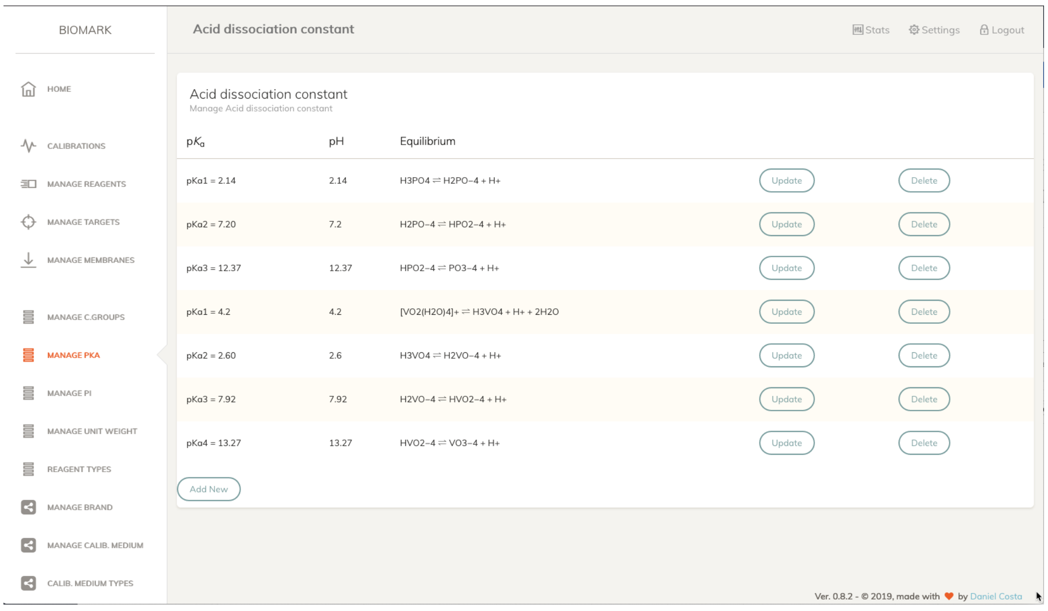
4.2.1 Fact-tables CRUD

To support the application relational data, there are many fact tables. A fact table in data warehousing consists of facts or data that commonly used but fairly changed. This kind of data usually have some additive values that supports the context in where it is used.

This web application currently have nine fact tables:

- Characteristic Group's;
- Acid dissociation constants;
- Isoelectric point's;
- Weight Units;
- Reagent Types;
- Brands;
- Calibration Mediums;
- Calibration Medium Types;
- Storage Types;

Each of this items has it's own Create, Read, Update, Delete (CRUD) interface. This user interfaces does not change in the work loggic, changing only on it's contents. Here is an example of the "Acid dissociation constants" List (figure 4.16).



Acid dissociation constant
Manage Acid dissociation constant

pK_a	pH	Equilibrium		
pKa1 = 2.14	2.14	H3PO4 ⇌ H2PO−4 + H+	Update	Delete
pKa2 = 7.20	7.2	H2PO−4 ⇌ HPO2−4 + H+	Update	Delete
pKa3 = 12.37	12.37	HPO2−4 ⇌ PO3−4 + H+	Update	Delete
pKa1 = 4.2	4.2	[VO2(H2O)4]+ ⇌ H3VO4 + H+ + 2H2O	Update	Delete
pKa2 = 2.60	2.6	H3VO4 ⇌ H2VO−4 + H+	Update	Delete
pKa3 = 7.92	7.92	H2VO−4 ⇌ HVO2−4 + H+	Update	Delete
pKa4 = 13.27	13.27	HVO2−4 ⇌ VO3−4 + H+	Update	Delete

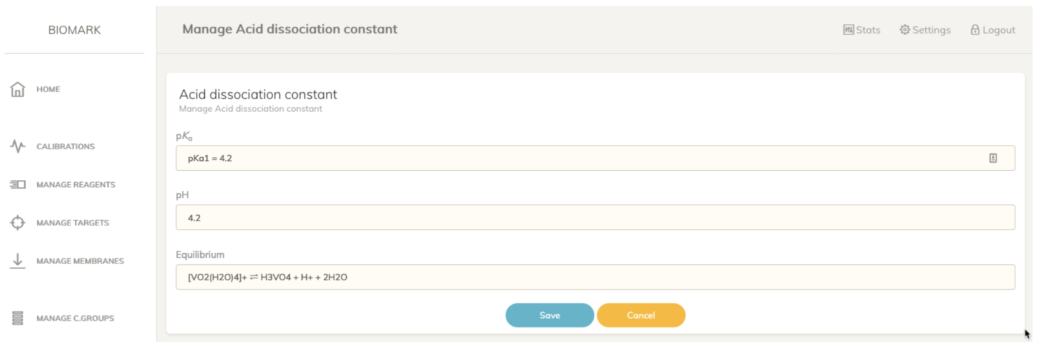
Add New

Ver. 0.8.2 - © 2019, made with ❤ by Daniel Costa

Figure 4.16: Example of a fact table list visualization.

In this page, the user is allowed to see the inserted data, delete line by line, add or update entries.

The ”Add New” and ”Update” buttons, uses the same form page, but, in the update scenarios it keeps the same entry Id, and shows the form filled with the edited entry (example of a filled item entry - figure 4.17)



Manage Acid dissociation constant

Acid dissociation constant
Manage Acid dissociation constant

pK_a
pKa1 = 4.2

pH
4.2

Equilibrium
[VO2(H2O)4]+ ⇌ H3VO4 + H+ + 2H2O

Figure 4.17: Add or Update form.

4.2.2 Calibration Box Management

This webapp was developed in order to manage multiple calibrations and CB at same time. To add or manage CB’s there is a special menu option. This is similar with the traditional CRUD (see 4.2.1) but it is relevant to describe in this section how to manage an existing CB.

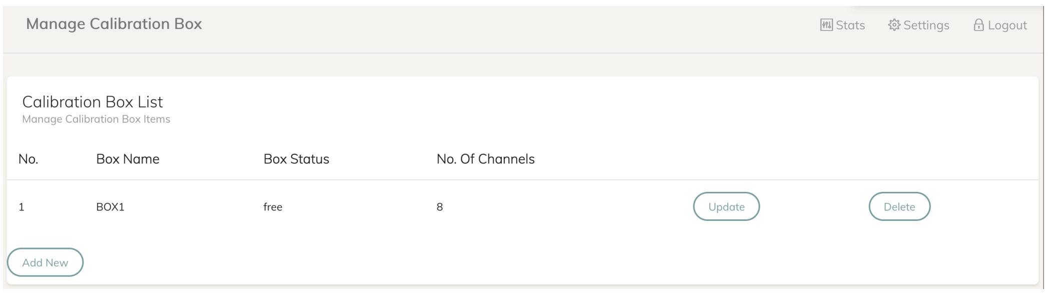
A CB, is non persistent memory device, made to automate a process and deliver their results. Sometimes, if there was a power or communication interruption, the CB status could stay in an invalid mode becoming impossible to assign new jobs.

There are another scenarios where a researcher needs to use the automatic pump for another research, and due to this the volume addition now should be configured as manual. All of this configurations should be done on the management page of the calibration box.

To find this page, the user should first login to the web application, and then click in the menu option called ”Manage Calib. Box” (figure 4.18).

Figure 4.18: Manage Calibration Box Menu Item.

In this list, similar with the fact tables list, the user is allowed to manage or add new CB.


Manage Calibration Box

Stats Settings Logout

Calibration Box List

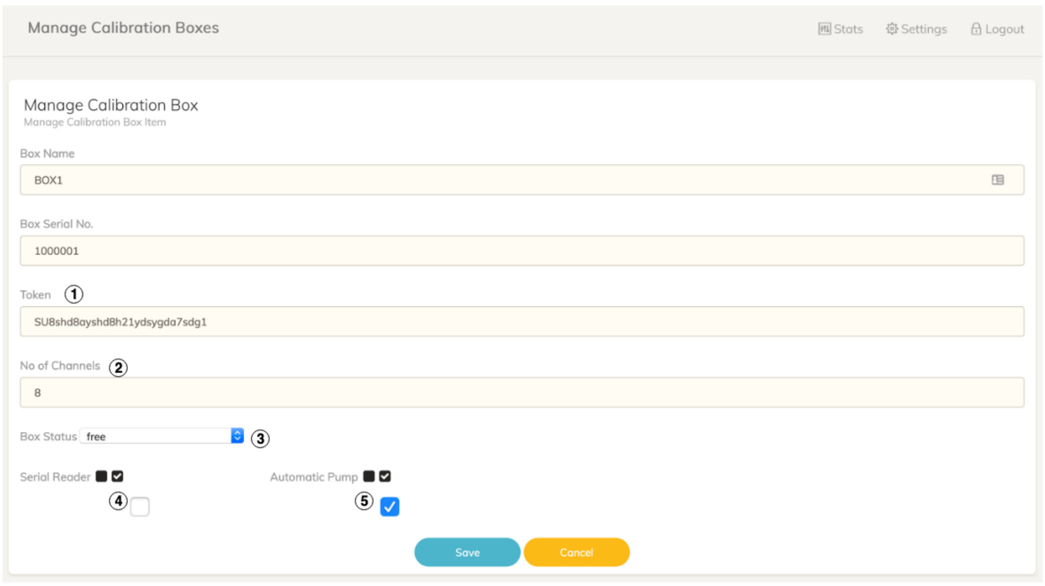
Manage Calibration Box Items

No.	Box Name	Box Status	No. Of Channels		
1	BOX1	free	8	Update	Delete

Add New

Figure 4.19: List of Calibration Box's.

To add a new CB, the user must provide its ”Box Name”, ”Box Serial Number”, ”The Box Authentication Token” (figure 4.20, 1), ”Number of channels” (figure 4.20, 2) available, if this CB has an automatic pump connected (figure 4.20, 5), and if there is a serial reader (pH Meter)(figure 4.20, 4), or, if not, the box will simulate values generating random values (useful for CB development and testing without Crison pH meter equipment). Using the status combo-box the user is allowed to reset the CB status (figure 4.20, 3), or define another that disables the CB.



Manage Calibration Boxes

Stats Settings Logout

Manage Calibration Box

Manage Calibration Box Item

Box Name	BOX1
Box Serial No.	1000001
Token ①	SU8shd8ayshd8h21ydsygda7sdg1
No of Channels ②	8
Box Status	free ③
Serial Reader ④	☐
Automatic Pump ⑤	☑

Figure 4.20: Managing Calibration Box.

4.2.3 Start Automatic Calibration

After prepare the CB and fill all the necessary fact tables, it is time to start an automatic calibration. All we need is available in the above menu (figure4.21).

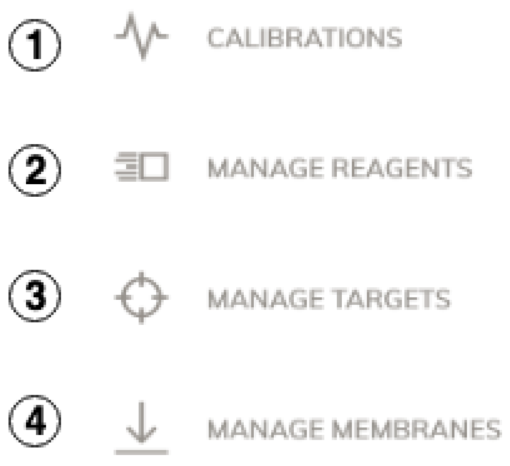
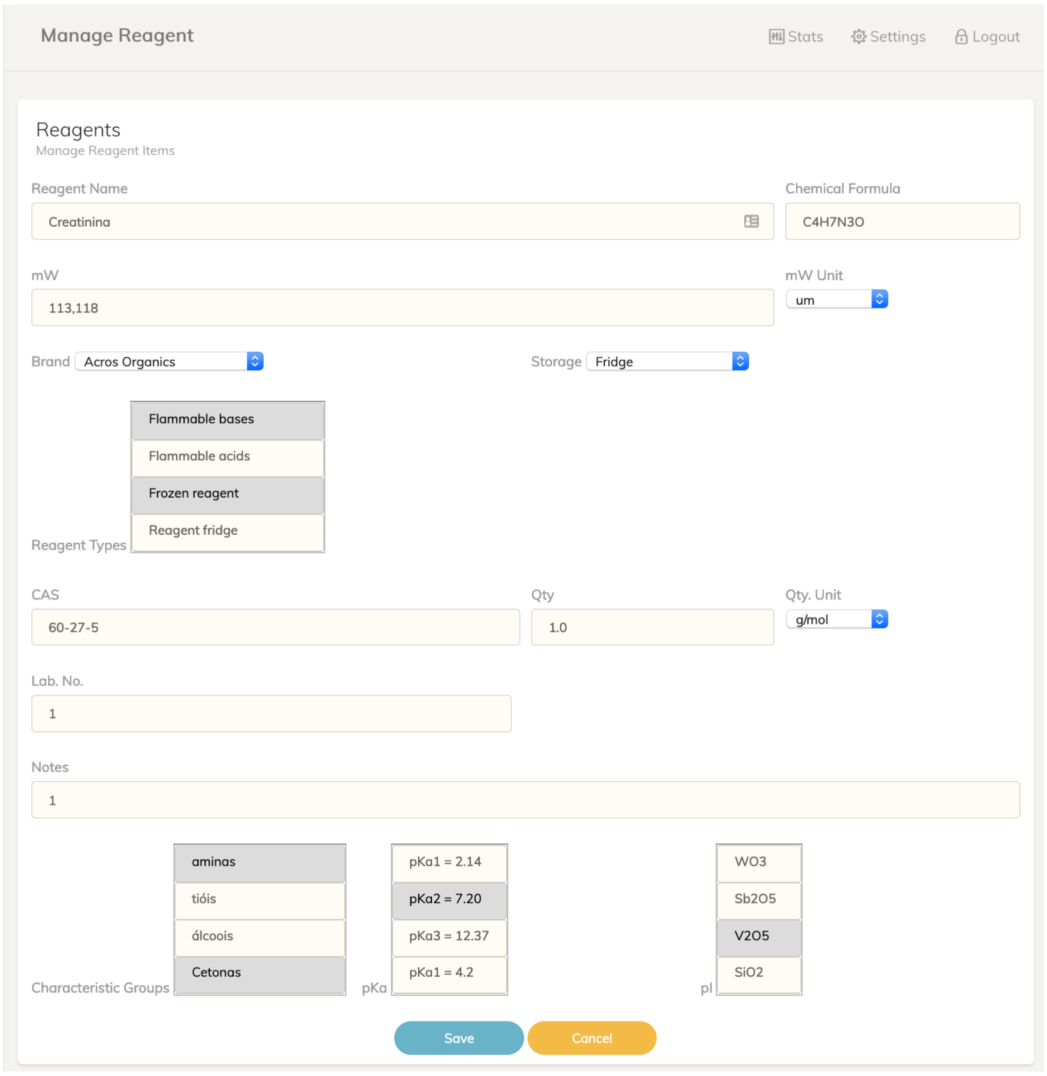


Figure 4.21: Calibration Menu.

First the user should manage the reagents used during the calibration clicking in the menu option ”Manage Reagents” (figure 4.21, 2).

The add or edit reagent page includes all the reagent relevant characteristics for the management and for the laboratory organization (figure 4.22).



Manage Reagent

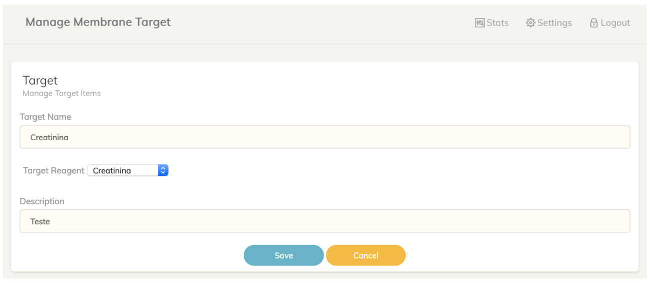
Reagents
Manage Reagent Items

Reagent Name	Creatinina
Chemical Formula	C4H7N3O
mW	113,118
mW Unit	um
Brand	Acros Organics
Storage	Fridge
Reagent Types	Flammable bases, Flammable acids, Frozen reagent, Reagent fridge
CAS	60-27-5
Qty	1.0
Qty. Unit	g/mol
Lab. No.	1
Notes	1
Characteristic Groups	aminas, tióis, álcoois, Cetonas
pKa	pKa1 = 2.14, pKa2 = 7.20, pKa3 = 12.37, pKa1 = 4.2
pI	WO3, Sb2O5, V2O5, SiO2

Figure 4.22: Edit/Add Reagent.

The fields ”Reagent Types”, ”Brand”, ”Storage”, ”Characteristic Groups”, ”pKa”, ”pI” are from the fact tables. ”Reagent Types”, ”Characteristic Groups”, ”pKa”,”pI” allow multi-selection items.

After adding the reagents evolved in the calibration, it is time to add the ”Membrane Target” (figure 4.21, 3 and figure 4.23) this is the target molecule. This target consists on its reagent, a reference name and a description.



Target
Manage Target Items

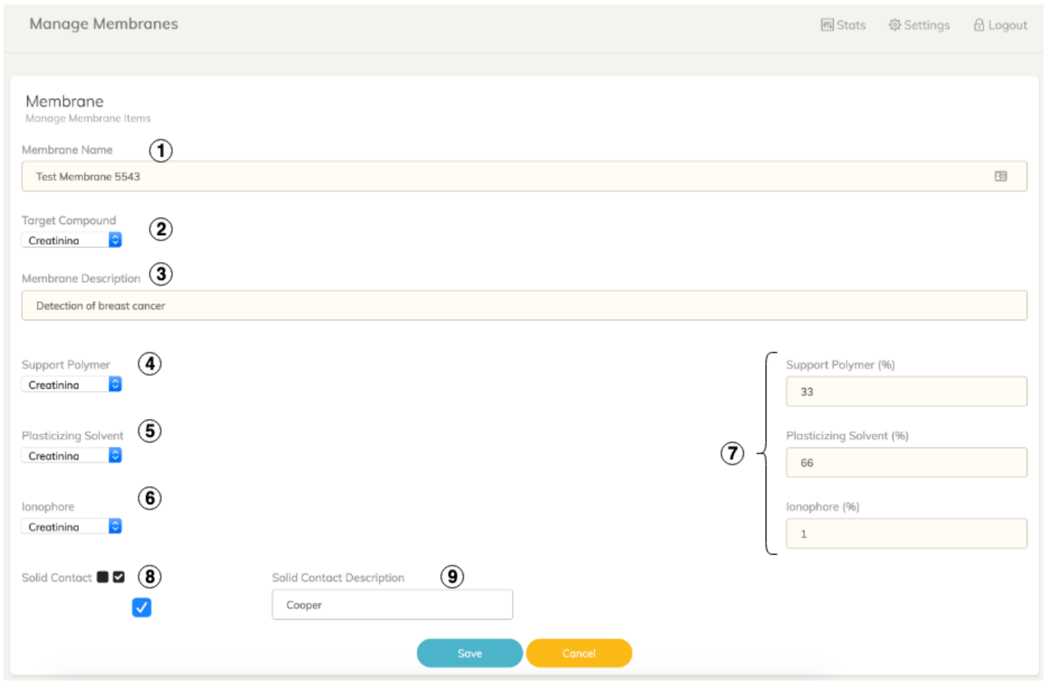
Target Name: Creatinina

Target Reagent: Creatinina

Description: Teste

Figure 4.23: Manage Membrane Target.

Then, it is the moment to add the membranes characteristics adding a ”Membrane” (figure 4.18, 4).



Membrane
Manage Membrane Items

Membrane Name: Test Membrane 5543

Target Compound: Creatinina

Membrane Description: Detection of breast cancer

Support Polymer: Creatinina

Plasticizing Solvent: Creatinina

Ionophore: Creatinina

Support Polymer (%): 33

Plasticizing Solvent (%): 66

Ionophore (%): 1

Solid Contact: ✓

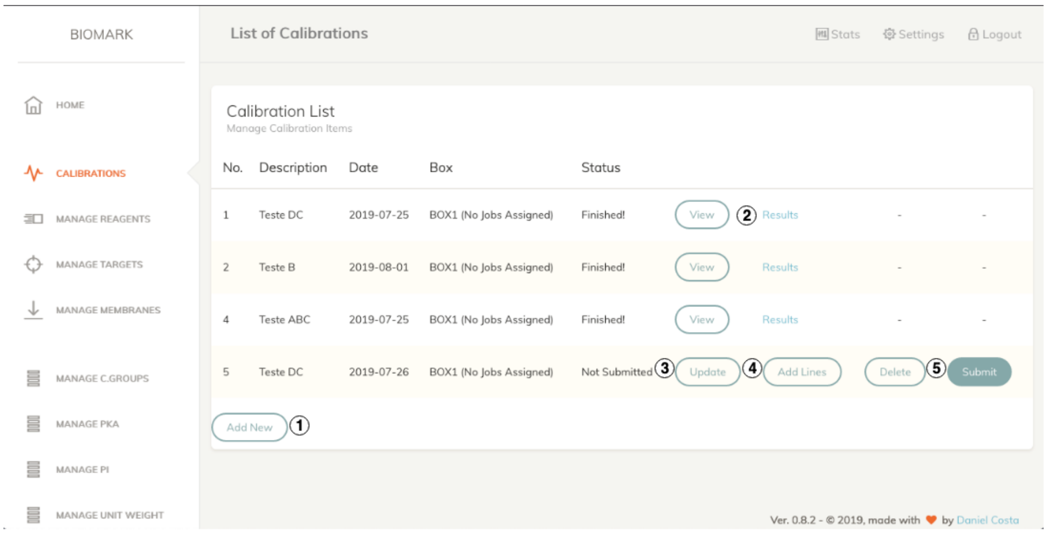
Solid Contact Description: Cooper

Figure 4.24: Manage Membrane.

To add or edit a membrane, the researcher need to first set the membrane name (figure 4.24, 1), this name, is set in the software and also in the membrane to physically identify. After this set the membrane target (figure 4.23), by selecting in a drop down box (figure 4.24, 2). Than the researcher adds a description note (figure 4.24, 3) and set the reagents (figure 4.22) compounds that are present in this membrane by setting the ”Support Polymer” (figure 4.24, 4), the ”Plasticizing Solvent” (figure 4.24, 5), the ”Ionophore” (figure 4.24, 6), and its percentage (figure 4.24, 7), and in case of there is a solid contact (figure 4.24, 8), the researcher describes this contact in the field ”Solid Contact Description” (figure 4.24, 9).

The researcher repeats this process for each membrane, reagent, target, in order to have the necessary items to configure and start a calibration.

Using the "Calibration" (figure 4.21, 1) menu option, the researcher will be able to see a list with all the configured Calibrations (figure 4.25), and also add new ones (figure 4.25, 1) or look for finished calibrations results (figure 4.25, 2).



Calibration List

Manage Calibration Items

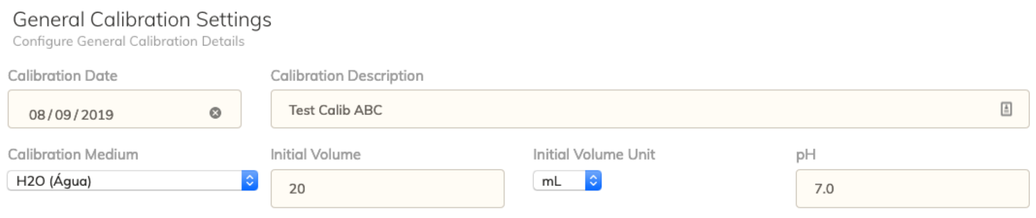
No.	Description	Date	Box	Status					
1	Teste DC	2019-07-25	BOX1 (No Jobs Assigned)	Finished!	View	2 Results		-	-
2	Teste B	2019-08-01	BOX1 (No Jobs Assigned)	Finished!	View	Results		-	-
4	Teste ABC	2019-07-25	BOX1 (No Jobs Assigned)	Finished!	View	Results		-	-
5	Teste DC	2019-07-26	BOX1 (No Jobs Assigned)	Not Submitted	3 Update	4 Add Lines		Delete	5 Submit

Ver. 0.8.2 - © 2019, made with ♥ by Daniel Costa

Figure 4.25: List of Calibrations

The add/view/edit calibration pages are the most complex pages on this project, it evolve a lot of relational fields, user experience in order to allow user only use the keyboard (TAB, ENTER) in order to enter values and configure the Calibration.

Dividing this page in sections the first section is the "General Calibration Settings" (figure 4.26) where the researcher configures the calibration date, it's name, the "Calibration Medium" and it's "Initial Volume" and the Medium pH.



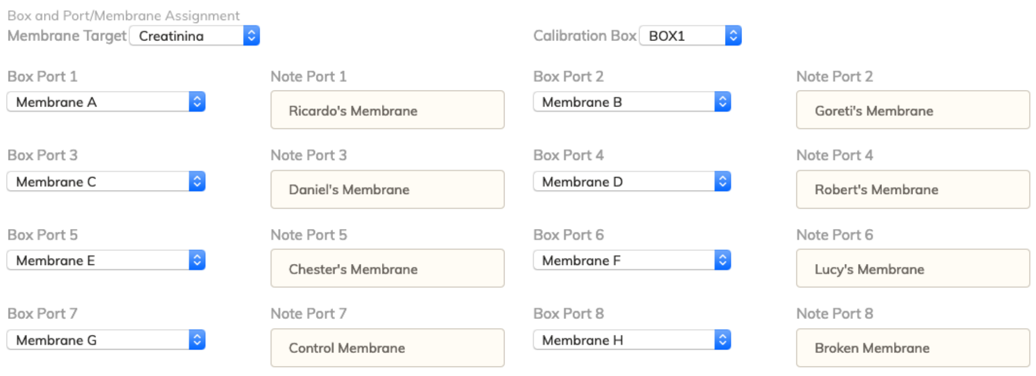
General Calibration Settings

Configure General Calibration Details

Calibration Date	08/09/2019
Calibration Description	Test Calib ABC
Calibration Medium	H2O (Água)
Initial Volume	20
Initial Volume Unit	mL
pH	7.0

Figure 4.26: Calibration General Settings

The researcher now needs to enter the pre-configured membranes (figure 4.24), target (figure 4.23), and the destination test Calibration Box (figure 4.18).

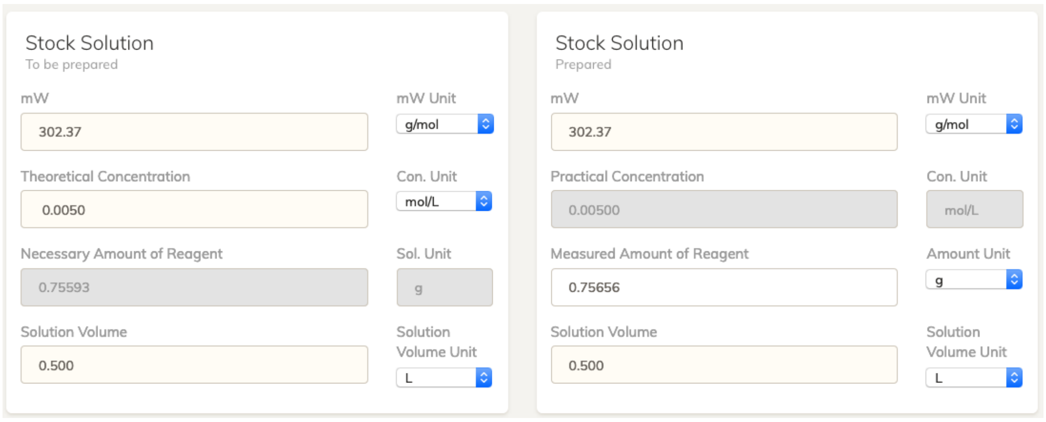


Box and Port/Membrane Assignment

Membrane Target: Creatinina		Calibration Box: BOX1	
Box Port 1: Membrane A	Note Port 1: Ricardo's Membrane	Box Port 2: Membrane B	Note Port 2: Goreti's Membrane
Box Port 3: Membrane C	Note Port 3: Daniel's Membrane	Box Port 4: Membrane D	Note Port 4: Robert's Membrane
Box Port 5: Membrane E	Note Port 5: Chester's Membrane	Box Port 6: Membrane F	Note Port 6: Lucy's Membrane
Box Port 7: Membrane G	Note Port 7: Control Membrane	Box Port 8: Membrane H	Note Port 8: Broken Membrane

Figure 4.27: Calibration Targets and Membranes

During the calibration process preparation, the researcher needs to calculate the "Stock Solution" to be added during the calibration process. This part is done in two step's, first the researcher plans the solution "To be prepared" where the main goal is to find the "Necessary Amount of Reagent" to achieve the "Theoretical Concentration" wanted, but, during the solution preparation, the researcher faces some impossible volume or weights for the preparation, and, due to this, the researcher needs to adjust it's preparation to the best possible alternative in order to achieve the possible "Practical Concentration" by configure the real values in the "Prepared" fields on the web application.



Stock Solution
To be prepared

mW: 302.37	mW Unit: g/mol
Theoretical Concentration: 0.0050	Con. Unit: mol/L
Necessary Amount of Reagent: 0.75593	Sol. Unit: g
Solution Volume: 0.500	Solution Volume Unit: L

Stock Solution
Prepared

mW: 302.37	mW Unit: g/mol
Practical Concentration: 0.00500	Con. Unit: mol/L
Measured Amount of Reagent: 0.75656	Amount Unit: g
Solution Volume: 0.500	Solution Volume Unit: L

Figure 4.28: Preparing Stock Solution

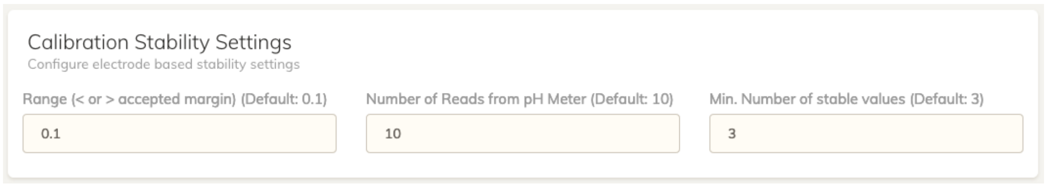
Calculations (rounded 5 decimal places):

Necessary Amount of Reagent = mW × Theoretical Concentration × Sol. Volume

$$\text{Practical Concentration} = \frac{MeasuredAmountofReagent}{mW \times SolutionVolume}$$

After the "General Calibration Settings" the researcher configures the "Calibration Stability Settings". During the manual calibration, the researcher keeps looking for the Crison pH meter values, waiting for the stability point where the variation of the value is less than an acceptable value. The researcher also knows that the membrane should have a stable result in a particular amount of time.

To configure this settings, the researcher should adjust the pre-configured settings:


Calibration Stability Settings
Configure electrode based stability settings

Range (< or > accepted margin) (Default: 0.1)	Number of Reads from pH Meter (Default: 10)	Min. Number of stable values (Default: 3)
0.1	10	3

Figure 4.29: Configure Calibration Stability Settings

In this settings there are three filed to be configured:

- **Range** - The accepted variability. This means that if there are at least the min. stable values with this variability (grater or lower then) the measurement is accepted;

- **Number of reads from pH Meter** - This is the number of measurements read from the pH Meter Serial port. If each read takes 1 second to me collected, this means that each measurement takes a maximum of 10 seconds (configurable) to occur, if there is no stable measurement during this 10 reads, the automation process proceeds to the next membrane;

- **Min. Number of stable values** - this is the number of values that should have a greater or lower variability then the configured in the range field. If there is a stable value, the calibration process continues to the next membrane and do not wait to complete the configured number of reads.

After submit a calibration configuration, the researcher have to add the addition of medium volume in the "Add Lines" button and page (figure 4.30).

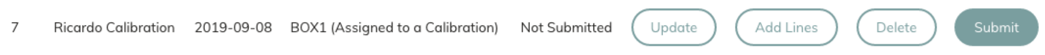
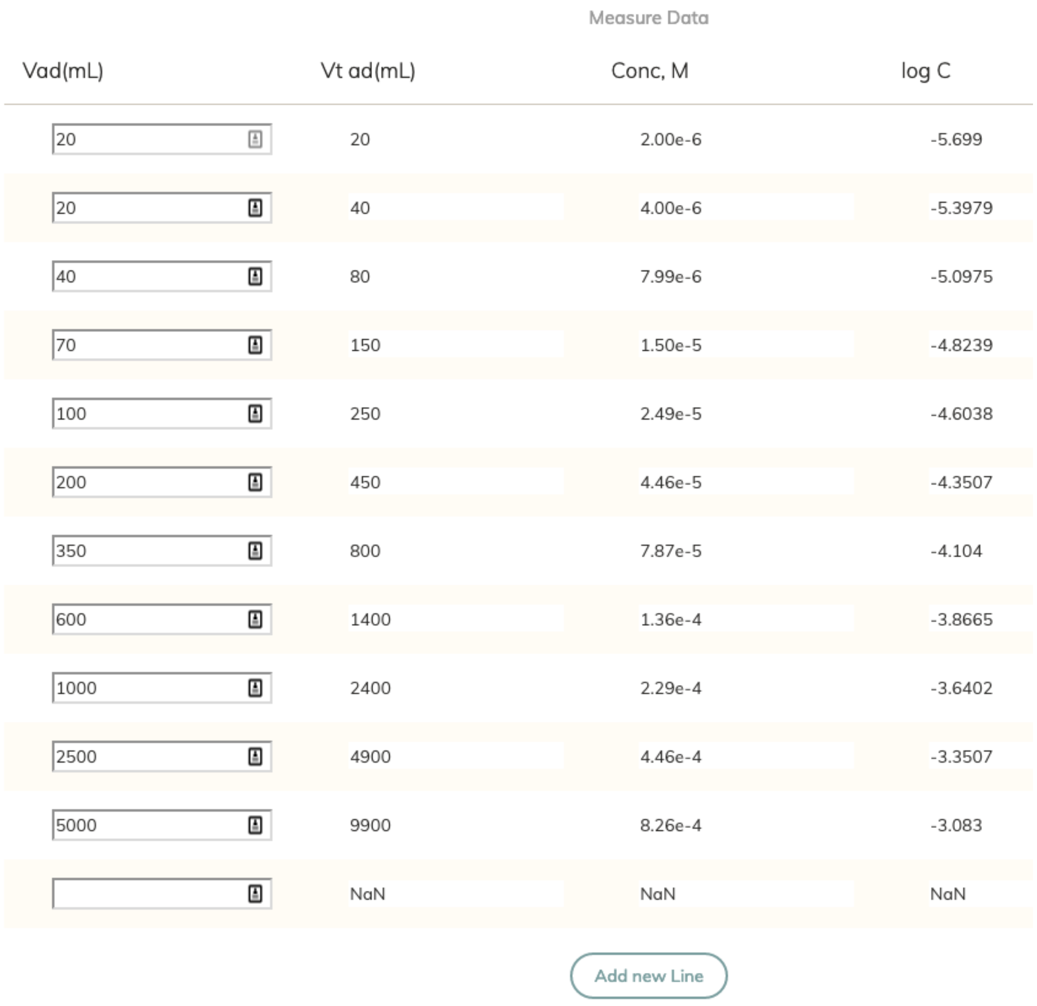

7	Ricardo Calibration	2019-09-08	BOX1 (Assigned to a Calibration)	Not Submitted	Update	Add Lines	Delete	Submit

Figure 4.30: Calibration Inserted - Add Lines Button

To add new volume addition lines (figure 4.31), the research simply enters the value of the additional volume, and presses the "ENTER" key. Another option is to click the "Add new Line" button. All the calculations are automatically done each time the researcher presses a key.



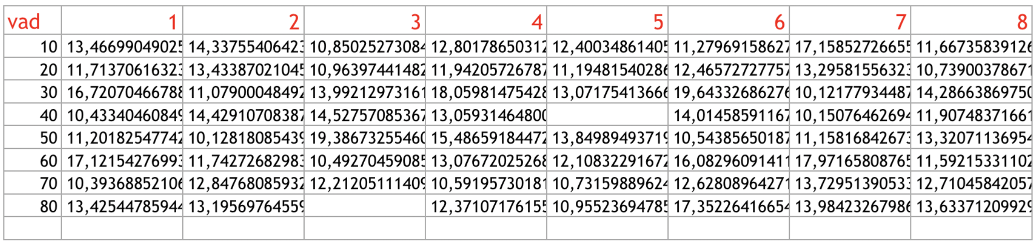
Vad(mL)	Vt ad(mL)	Conc, M	log C
20	20	2.00e-6	-5.699
20	40	4.00e-6	-5.3979
40	80	7.99e-6	-5.0975
70	150	1.50e-5	-4.8239
100	250	2.49e-5	-4.6038
200	450	4.46e-5	-4.3507
350	800	7.87e-5	-4.104
600	1400	1.36e-4	-3.8665
1000	2400	2.29e-4	-3.6402
2500	4900	4.46e-4	-3.3507
5000	9900	8.26e-4	-3.083
	NaN	NaN	NaN

Figure 4.31: Calibration - Add Volume Lines

This lines are the ones that the Calibration Box will use to automate the pump volume addition every time the calibration of all membranes are done.

In the end of all the software configuration and physical preparation, the researcher presses the ”Submit” button (figure 4.30) to start the automated calibration process.

If there is an automatic pump, the researcher just have to wait until the fully automated process finishes to download the excel file with the measured values (figure 4.32). If this calibration is made without an automatic pump, the researcher should pay attention each time the button led flashes to manually add the specified value present on the CB LCD.



vad	1	2	3	4	5	6	7	8
10	13,46699049025	14,33755406423	10,85025273084	12,80178650312	12,40034861405	11,27969158627	17,15852726655	11,66735839126
20	11,71370616323	13,43387021045	10,96397441482	11,94205726787	11,19481540286	12,46572727757	13,29581556323	10,73900378671
30	16,72070466788	11,07900048492	13,99212973161	18,05981475428	13,07175413666	19,64332686276	10,12177934487	14,28663869750
40	10,43340460849	14,42910708387	14,52757085367	13,05931464800		14,01458591167	10,15076462694	11,90748371661
50	11,20182547742	10,12818085439	19,38673255460	15,48659184472	13,84989493719	10,54385650187	11,15816842673	13,32071136954
60	17,12154276993	11,74272682983	10,49270459085	13,07672025268	12,10832291672	16,08296091411	17,97165808765	11,59215331102
70	10,39368852106	12,84768085932	12,21205111409	10,59195730181	10,73159889624	12,62808964271	13,72951390533	12,71045842057
80	13,42544785944	13,19569764559		12,37107176155	10,95523694785	17,35226416654	13,98423267986	13,63371209929

Figure 4.32: Calibration - Excel File with measured calibration values.

With this excel file, the researcher is able to do it's calculations and take their conclusions to prepare the next calibration with new adjustments.

Chapter 5

Calibration Box - Hardware/Software

After the appropriate architecture definition (figure 3.5), the development of the first Calibration Box prototype was a hand-made effort divided into the following steps:

- **5.1** - Hardware Costs;
- **5.2.1** - Structure and brackets;
- **5.2.2** - Input/Output connections;
- **5.2.3** - Power distribution;
- **5.2.4** - Modules assembly;
- **5.2.5** - Calibration Box operating system;
- **5.2.7** - Calibration Box software;
- **5.2.6** - Testing Calibration Box;
- **5.3.1** - Operating the Calibration Box;

5.1 Hardware Costs

All the hardware used to develop this CB prototype is open source hardware. Besides the development of the hardware is open, there are existing modules, cables and connectors that can be bought to assembly the first prototype.

Hardware prototype costs:

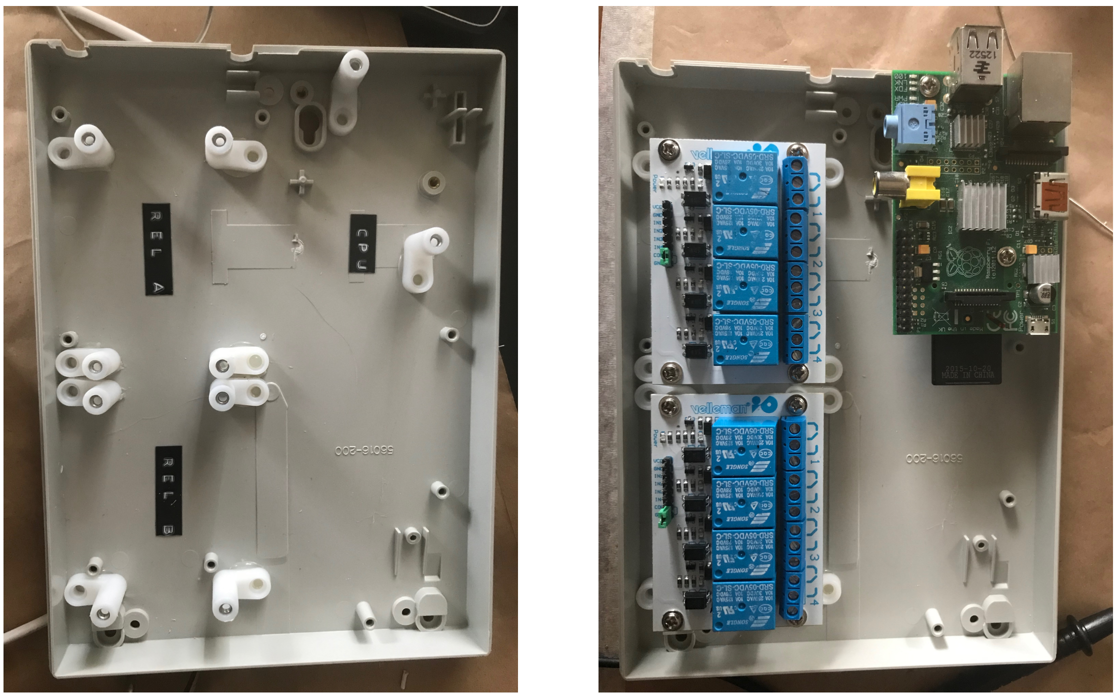
Item	**Description**	**Unit Price**	**Qty.**	**Total Price**
Computer	Raspberry Pi Model 4 1 Gb	39,90 €	1	39,90 €
USB-Serial	Ewent USB to Serial Adaptor	14,90 €	1	14,90 €
LCD	Display LCD 20x4 I2C	14,70 €	1	14,70 €
Power Supply	5.1V 2.5A MicroUSB	12,90 €	1	12,90 €
8 Port Relays	8 Port Optocopler Relay	6,94 €	1	6,94 €
Button w/ LED	16 mm Illuminated Button	5,97 €	1	5,97 €
SD Card	Micro SD Card	5,90 €	1	5,90 €
RCA Connectors	RCA Connectors	0,70 €	8	5,60 €
Patch Cables	Patch Cables	5,00 €	1	5,00 €
Box	Telecom Box	1,00 €	1	1,00 €
BNC Connector	BNC Connector	1,00 €	1	1,00 €
Coax Cable	Coax Cable	1,00 €	1	1,00 €
Total				**114,81 €**

Table 5.1: Hardware prototype costs.

5.2 Inside Box

5.2.1 Structure and brackets

The prototype model was made based on a telecommunications plastic box. Since this box is not specifically made for this prototype assembly, there were some necessary changes to be implemented to add the necessary modules brackets.



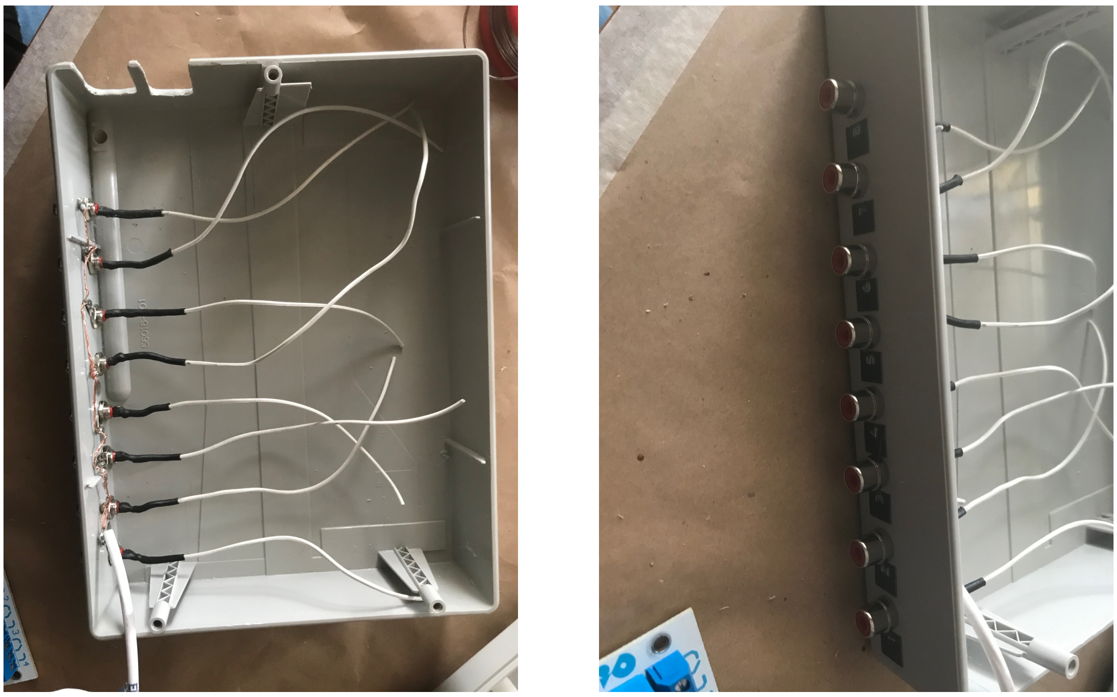
(a) Before mount (b) After mount

Figure 5.1: Hardware - Module assembly brackets.

Figure 5.1 - it is possible to see on the right, two modules and the CPU assembly brackets (a) . Each module location has their specific identification label (a). On the left it is possible to see the modules and the CPU mounted with screws.

5.2.2 Input/Output connections

To provide the necessary input and output connections for the CB, there wall mounted connectors wore assembled in hand-made holes.



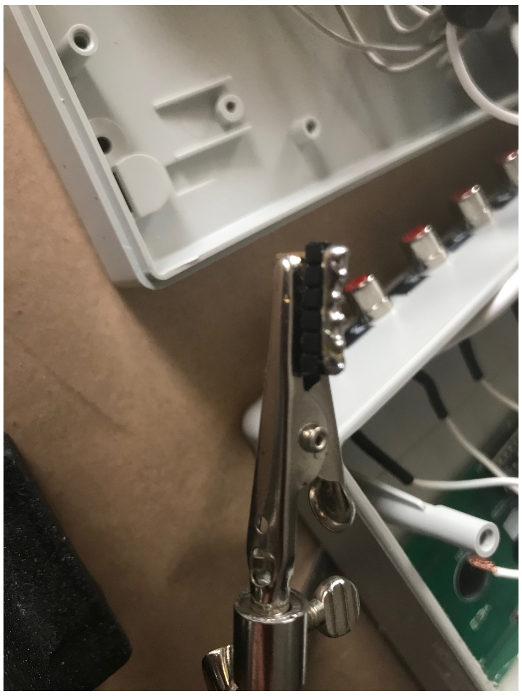
(a) Inside View (b) Outside View

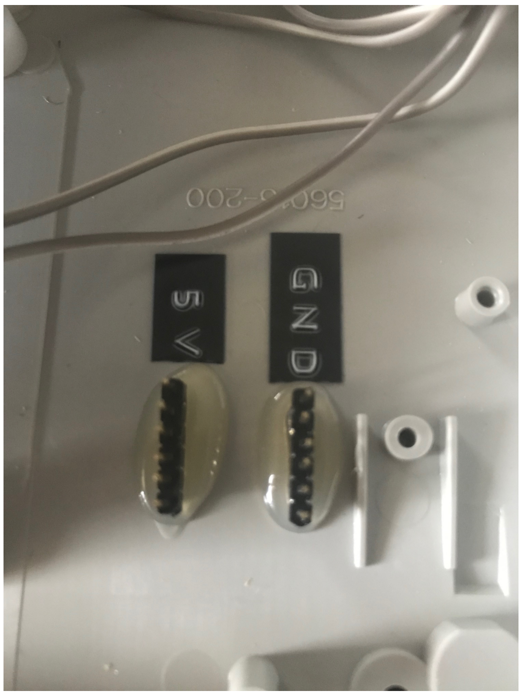
Figure 5.2: Hardware - Input Output Connections.

Figure 5.2 - is it possible to see the RCA (b) connections with cooper cables and therm-retractable cable tube (a). It is also possible to see the Raspberry Pi ports cut's on the plastic box (a).

5.2.3 Power distribution

The Raspberry Pi power source (USB Power Supply) is the only power source to the CB. The connected modules need 5V power-source. To distribute this power-source, there was attached to the box the necessary in serial pins.


(a) Pins distribution



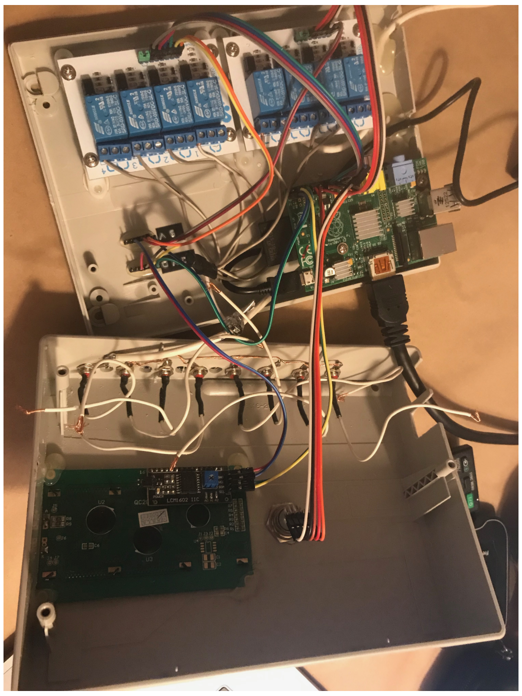
(b) Labeled power distributors

Figure 5.3: Hardware - 5V power distribution and GND

Figure 5.3 - the power distribution made via soldered pins (a) glued to the box with descriptive labels (b).

5.2.4 Modules assembly

After mounting the modules and the input output ports, with the power distribution in place, the module interconnection with the main CPU was achieved using patch cables connected with the GPIO interface and power distribution.


(a) Modules Connected

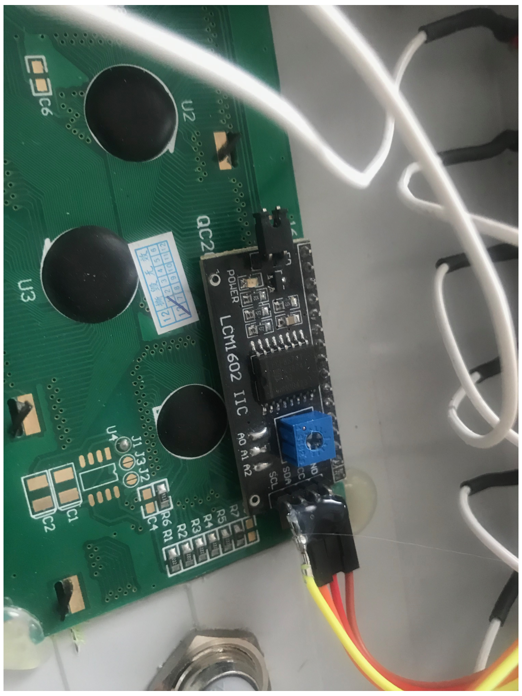

(b) I2C LCD Connection

Figure 5.4: Hardware - Modules connected.

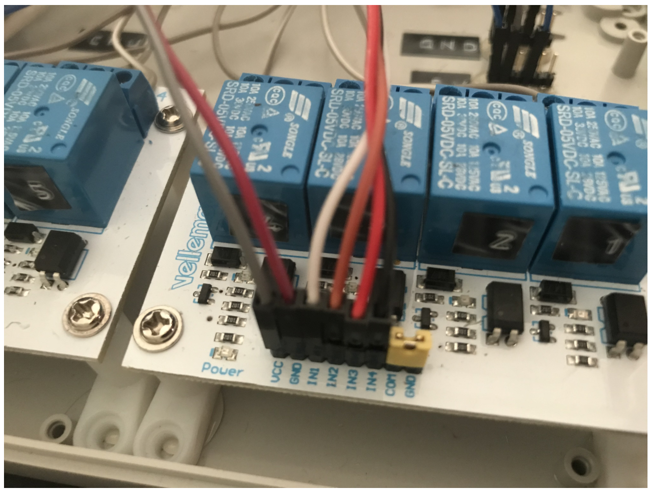


Figure 5.5: Relays connection

5.2.5 Calibration Box operating system

Since the CB has a computer that runs a *Java Application*, the chosen operating system was the *Debian GNU/Linux Operating System*, particularly, *Raspbian* (a Debian [33] adaptation for the Raspberry Pi [34]).

The operating system installation process can be found in the project reference URL [35].

After prepare the Raspbian SD Card 5.6, the card should be installed on the Raspberry Pi and the initial configuration process should start.

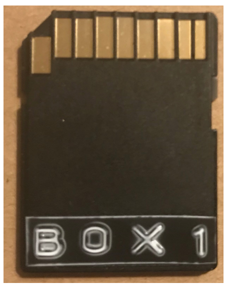


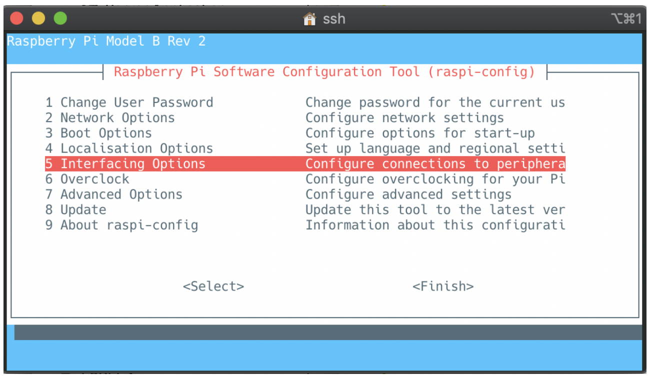
Figure 5.6: Operating System SD Card.

To start, the following configuration is needed to support the I2C LCD display:

Run:

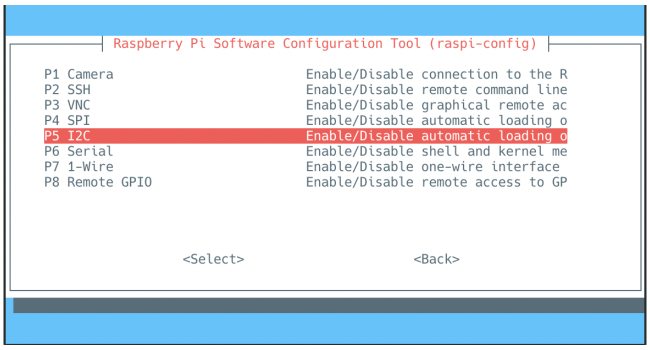
```
sudo raspi-config
```

In the menu, follow open ”Interfacing Options”:



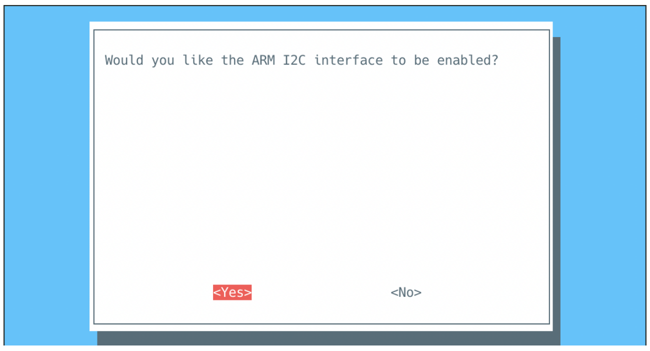
```
Raspberry Pi Model B Rev 2
            Raspberry Pi Software Configuration Tool (raspi-config)

 1 Change User Password          Change password for the current us
 2 Network Options               Configure network settings
 3 Boot Options                  Configure options for start-up
 4 Localisation Options          Set up language and regional setti
 5 Interfacing Options           Configure connections to periphera
 6 Overclock                     Configure overclocking for your Pi
 7 Advanced Options              Configure advanced settings
 8 Update                        Update this tool to the latest ver
 9 About raspi-config            Information about this configurati

                 <Select>                    <Finish>
```

Figure 5.7: raspi-config: Interfacing Options.



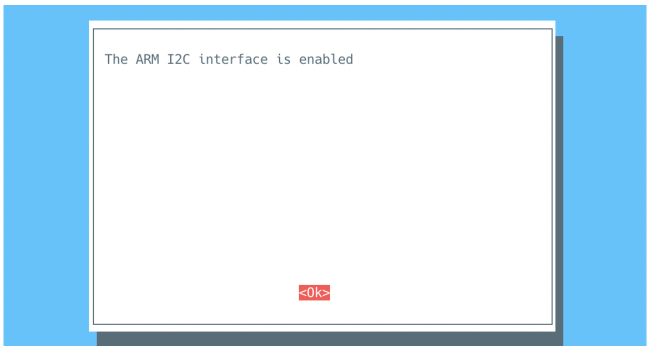
```
Raspberry Pi Software Configuration Tool (raspi-config)

P1 Camera                         Enable/Disable connection to the R
P2 SSH                            Enable/Disable remote command line
P3 VNC                            Enable/Disable graphical remote ac
P4 SPI                            Enable/Disable automatic loading o
P5 I2C                            Enable/Disable automatic loading o
P6 Serial                         Enable/Disable shell and kernel me
P7 1-Wire                         Enable/Disable one-wire interface
P8 Remote GPIO                    Enable/Disable remote access to GP

                 <Select>                     <Back>
```

Figure 5.8: raspi-config: Enable I2C.



```
Would you like the ARM I2C interface to be enabled?

                 <Yes>                  <No>
```

Figure 5.9: raspi-config: Enable I2C - Enable Option.



```
The ARM I2C interface is enabled

                          <Ok>
```

Figure 5.10: raspi-config: Enable I2C - Confirmation.

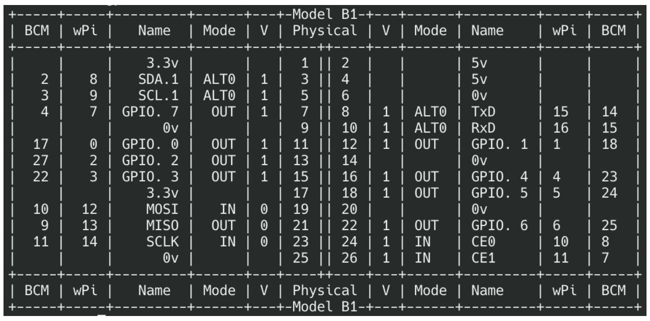
In the same configuration menu the user is also able to configure the network settings, and enable remote management via SSH. Since this CB connects with the cloud based application, and the default network configuration uses DHCP, if the connected network allows direct access to the Internet, this CB should be able to work normally.

The following packages should be installed in order to allow Java interaction with the GPIO using Pi4J Library [36].

```
sudo apt-get install python-smbus
sudo apt-get install i2c-tools
sudo usermod -a -G gpio pi
```

After reboot, all the conditions to run the CB software should be ready, but, to configure the GPIO pins in the Java application, the following command allow us to see the available pins and numbers 5.11.

```
gpio readall
```



```
+-----+-----+---------+------+---+-Model B1-+---+------+---------+-----+-----+
| BCM | wPi |   Name  | Mode | V | Physical | V | Mode | Name    | wPi | BCM |
+-----+-----+---------+------+---+----++----+---+------+---------+-----+-----+
|     |     |    3.3v |      |   |  1 || 2  |   |      | 5v      |     |     |
|   2 |   8 |   SDA.1 | ALT0 | 1 |  3 || 4  |   |      | 5v      |     |     |
|   3 |   9 |   SCL.1 | ALT0 | 1 |  5 || 6  |   |      | 0v      |     |     |
|   4 |   7 | GPIO. 7 |  OUT | 1 |  7 || 8  | 1 | ALT0 | TxD     | 15  | 14  |
|     |     |      0v |      |   |  9 || 10 | 1 | ALT0 | RxD     | 16  | 15  |
|  17 |   0 | GPIO. 0 |  OUT | 1 | 11 || 12 | 1 | OUT  | GPIO. 1 | 1   | 18  |
|  27 |   2 | GPIO. 2 |  OUT | 1 | 13 || 14 |   |      | 0v      |     |     |
|  22 |   3 | GPIO. 3 |  OUT | 1 | 15 || 16 | 1 | OUT  | GPIO. 4 | 4   | 23  |
|     |     |    3.3v |      |   | 17 || 18 | 1 | OUT  | GPIO. 5 | 5   | 24  |
|  10 |  12 |    MOSI |   IN | 0 | 19 || 20 |   |      | 0v      |     |     |
|   9 |  13 |    MISO |  OUT | 0 | 21 || 22 | 1 | OUT  | GPIO. 6 | 6   | 25  |
|  11 |  14 |    SCLK |   IN | 0 | 23 || 24 | 1 | IN   | CE0     | 10  | 8   |
|     |     |      0v |      |   | 25 || 26 | 1 | IN   | CE1     | 11  | 7   |
+-----+-----+---------+------+---+----++----+---+------+---------+-----+-----+
| BCM | wPi |   Name  | Mode | V | Physical | V | Mode | Name    | wPi | BCM |
+-----+-----+---------+------+---+-Model B1-+---+------+---------+-----+-----+
```

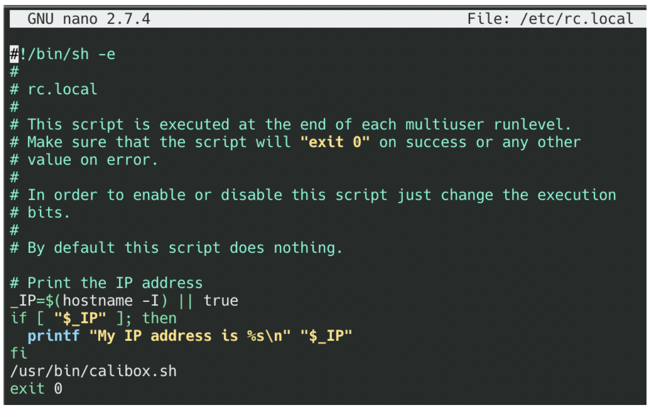
Figure 5.11: ”gpio readall” command output. Display GPIO pin numbers.

The last step is to add the Java application to the startup by copying the application jar to: ”/usr/bin/calibox.jar”, adding the following startup script at ”/usr/bin/calibox.sh”.

```
#!/bin/bash
java -jar /usr/bin/calibox.jar
```

And then set this script to be executed after operating system starts editing the file ”/etc/rc.local” and adding the following line before the ”exit 0” line.

```
/usr/bin/calibox.sh
```

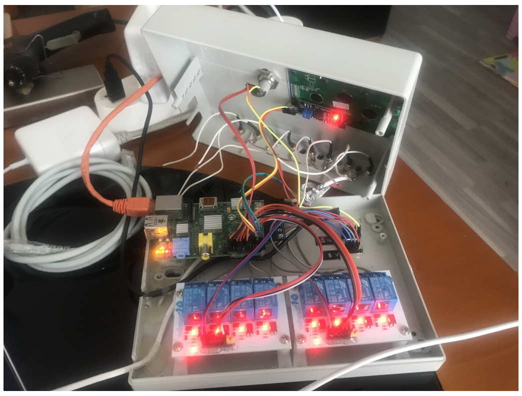


```
  GNU nano 2.7.4                                        File: /etc/rc.local

#!/bin/sh -e
#
# rc.local
#
# This script is executed at the end of each multiuser runlevel.
# Make sure that the script will "exit 0" on success or any other
# value on error.
#
# In order to enable or disable this script just change the execution
# bits.
#
# By default this script does nothing.

# Print the IP address
_IP=$(hostname -I) || true
if [ "$_IP" ]; then
  printf "My IP address is %s\n" "$_IP"
fi
/usr/bin/calibox.sh
exit 0
```

Figure 5.12: Full startup script.

5.2.6 Testing Calibration Box hardware and software

After the Calibration Box (CB) prototype assembly and the operating system installed, the strategy used to test the software/hardware interaction was developing small programs to control and test each component module. This methodology allowed the appropriate testing of each component before adding the business logic, and this experimental stage also contribute to the appropriate software development solution design.


(a) Relay Testing

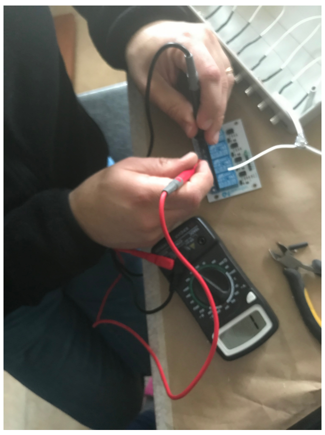

(b) Finding Relay Defaults

Figure 5.13: Hardware - Testing relays interaction.

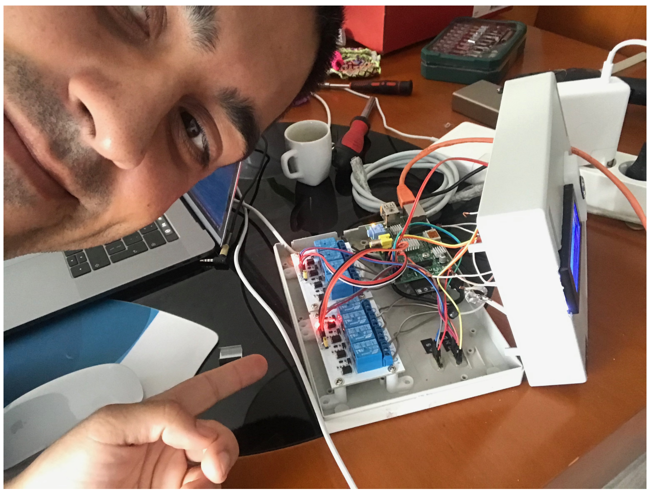


Figure 5.14: Testing all Hardware together.

In the figure 5.13 (b) it is possible to see the relay default behaviour by testing with a multimeter the interruption between the relay connections with or without enabling the relay action. The figure 5.13 (a), it is possible to see all the relays enabled, testing the Raspberry Pi GPIO pins defined in the architecture. In the figure 5.14, it is possible to see last hardware test with all the components and modules working together.

5.2.7 Calibration Box software

Like the server-side application 4.1.1, the client-side was implemented using OpenJDK 8 [16] also using Apache Maven [17].

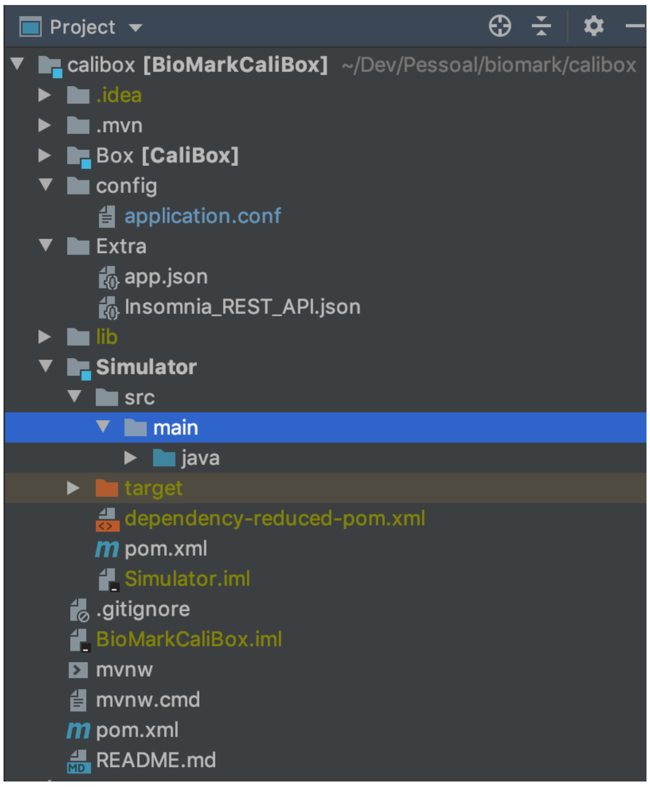
The following dependencies was used to develop the CB application:

Dependency	Version	License
resteasy-client	3.8.0.Final	Apache License, Version 2.0 [18]
resteasy-jackson2-provider	3.8.0.Final	Apache License, Version 2.0 [18]
resteasy-jackson-provider	3.8.0.Final	Apache License, Version 2.0 [18]
resteasy-multipart-provider	3.8.0.Final	Apache License, Version 2.0 [18]
commons-io	2.0.1	Apache License, Version 2.0 [18]
lombok	1.18.4	Apache License, Version 2.0 [18]
jcl-over-slf4j	1.7.12	MIT License [22]
logback-classic	1.1.3	Eclipse Public License - v 1.0 [19]
com.typesafe.config	1.3.3	Apache License, Version 2.0 [18]
poi-ooxml	3.17	Apache License, Version 2.0 [18]
pi4j-core	1.2	GNU General LGPL version 3.0 [37]
pi4j-device	1.2	GNU General LGPL version 3.0 [37]

Table 5.2: Calibration Box Dependencies and Licenses

The Java development using a computer different than the Raspberry Pi using the Pi4J library disallow the developer to test the business logic without deploying to the Raspberry Pi. To allow the appropriate development and testing, two maven modules were created, the ”simulator” and the ”box” modules.

The ”simulator” and the ”box” modules shares the same code and logic but, the ”simulator” does not include all the hardware interfacing and due to this it is possible to test without using the Raspberry Pi.



```
Project
▼ calibox [BioMarkCaliBox] ~/Dev/Pessoal/biomark/calibox
  ▶ .idea
  ▶ .mvn
  ▶ Box [CaliBox]
  ▼ config
      application.conf
  ▼ Extra
      app.json
      Insomnia_REST_API.json
  ▶ lib
  ▼ Simulator
    ▼ src
      ▼ main
        ▶ java
    ▶ target
      dependency-reduced-pom.xml
      pom.xml
      Simulator.iml
    .gitignore
    BioMarkCaliBox.iml
    mvnw
    mvnw.cmd
    pom.xml
    README.md
```

Figure 5.15: Calibration Box software project structure.

The code structure was made as defined during the architecture 3.1.2. But, there are some relevant code extracts that should be should be shared with this report.

The interaction with a researcher is made mainly using the button led and the LCD screen. This screen has the physical limit of 4 lines and 20 characters per line. To handle this limitation, and keep the displayed text clean the following code was written:

```java
public class LCDActions {
    private int no_char_per_line = 20;
    private int no_of_lines = 4;

    public void printToLCD(String text, int line){
        if(text != null && line >=0 && line <=(no_of_lines -1)) {
            clearLine(line);
            if(text.length() > no_char_per_line){
                text = text.substring(0, (no_char_per_line -1));
                log.warn("LCD Line with more than available
                   chars was trimmed! Line: " + line + ",
                   Trimmed text: '" + text + "'.");
            }
            RPi_HW.LCD.write(line, text);
            log.info("Print to LCD at Line " + line + " '" +
               text + "'");
        }else{
            log.warn("Tried to write to LCD to invalid location
               or empty text. Text: '" + text + "', line number:
               " + line);
        }
    }
    public void clearLine(int line){
        if(line >=0 && line <=(no_of_lines -1)) {
            RPi_HW.LCD.clear(line);
            log.info("Line " + line + " cleared!");
        }else{
            log.warn("Invalid line number to clear!");
        }
    }
}
```

To use this class writing a line to the LCD:

```
lcdActions.printToLCD("Box Status: " +
   caliBox.getStatus().getCaliboxStatus(), 3);
```

The user is able to set their own perspective of membrane stability, adjusting the settings in the web interface 4.29. This configuration is collected with the Job definition and the following code implements it:

```
public static Double measureStabilizer(List<Double> measures,
   Double acceptedVariability, int minStableValues){
    List<Double> stableMeasures = new ArrayList<>();
    if(measures.size() > 0 && measures.size() >=
       minStableValues){
        int i = 0;
        for (Double measure: measures) {
            if (!measure.isNaN()) {
                if (
                stableMeasures.size() < 1
                || ((measure + acceptedVariability) <=
                   stableMeasures.get(i - 1) && (measure +
```

```java
                        acceptedVariability) >= stableMeasures.get(i
                        - 1))
                    || ((measure - acceptedVariability) <=
                        stableMeasures.get(i - 1) && (measure -
                        acceptedVariability) <= stableMeasures.get(i
                        - 1))
                    ) {
                        stableMeasures.add(measure);
                        i++;
                    } else {
                        stableMeasures.clear();
                        stableMeasures.add(measure);
                        i = 1;
                    }
                if (stableMeasures.size() >= minStableValues)
                    return stableMeasures.get(stableMeasures.size()
                        - 1);
                }
            }
        }
    return Double.NaN;
}
```

In the above extract of code, the researcher configuration is applied by iterating with all the serial input reads and comparing the sequence of reads. If there is the ”stability point” this function returns the last stable value.

The final code extract is the CB port switcher, controlling all the relays, opening the connection between Crison pH meter and the selected membrane.

```java
public class Actions {
    public static boolean relayControl(GpioPinDigitalOutput pin,
        boolean on){
        if(on)
            pin.high();
        else
            pin.low();
        return on;
    }
    public static boolean
```

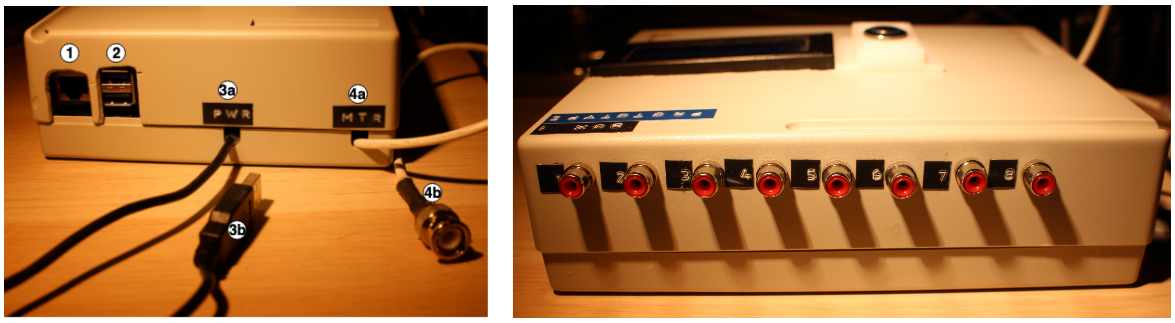
```
        relaySwitcher(List<GpioPinDigitalOutput> pins, int toPin){
        int i = 0;
        if(relaysSwitchOff(pins)) {
            relayControl(pins.get(toPin), true);
            log.info("Switched to relay " + (toPin + 1) + "!");
            log.debug("Relay " + (i + 1) + " turned on!");
            return true;
        }else{
            return false;
        }
    }
    public static boolean
        relaysSwitchOff(List<GpioPinDigitalOutput> pins){
        int i = 0;
        for (GpioPinDigitalOutput pin: pins) {
            log.debug("Relay " + (i + 1) + " turned off!");
            relayControl(pin, false);
            i++;
        }
        log.info("All Relays Switched Off!");
        return true;
    }
}
```

In the above code extract, all the relays are closed before opening the appropriate membrane connection relay preventing the surcharge of Crison pH meter with two open circuits.

5.3 Outside Box

Outside the Calibration Box, in the developed prototype, named ”BOX 1”, there are available the following interfaces:

1. One network interface(figure 5.16, a-1);
2. Two USB ports (figure 5.16, a-2):
    (a) One used for the USB-Serial adaptor to connect with Crison pH serial interface;
    (b) Another used to connect with the Legato 100 pump;
3. A USB (PWR) power source cable (figure 5.16, a-3a, a-3b);
4. One BNC (MTR) port to connect with the Crison pH Meter Input (figure 5.16, a-4a, a-4b);
5. Eight RCA ports to connect with the membranes (numbered figure 5.16, b);
6. One LCD (4x20) backlighted display (figure 5.16, b);
7. One click-button with blue LED (figure 5.16, b);



(a) Box System Interfaces (b) Box Membrane Interfaces

Figure 5.16: Calibration Box (CB) interfaces.

p° bh

5.3.1 Operating the Calibration Box

Currently there are two modes:

1. With automatic pump;
2. Without automatic pump;

The difference between those two modes is that the first mode does not require the user interaction after prepare the calibration and submit.

When the CB is in "free status" that means that the CB is ready to start a new calibration (figure 5.17).

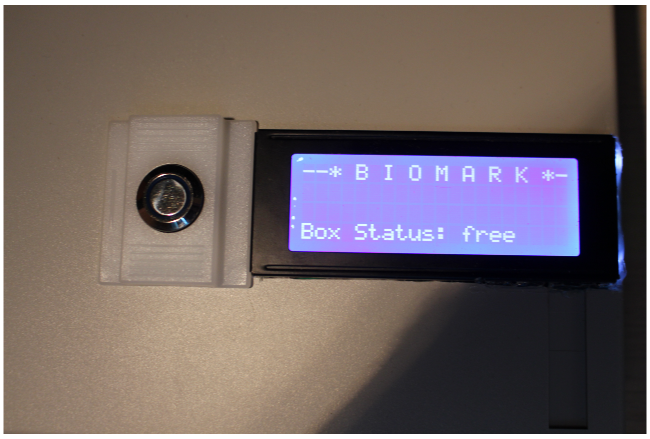


Figure 5.17: Calibration Box in free status.

After submit the calibration by pressing its button in the web interface (figure 4.30), the calibration starts. During the the execution the CB changes its status to "working" and starts reading values from the configured interfaces (figure 5.18).

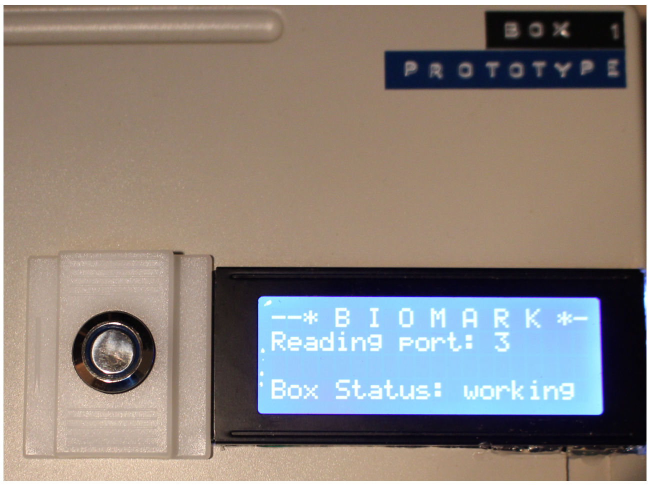


Figure 5.18: Calibration Box working. Reading from port 3.

In this stage of the calibration the CB measures from any configured membrane collecting all the possible reads from Crison pH meter, and looking for stable values. After read from all the membranes, if there is a Legato 100 automatic pump, the CB proceeds with the calibration by adding the amount of reagent configured (figure 5.19).

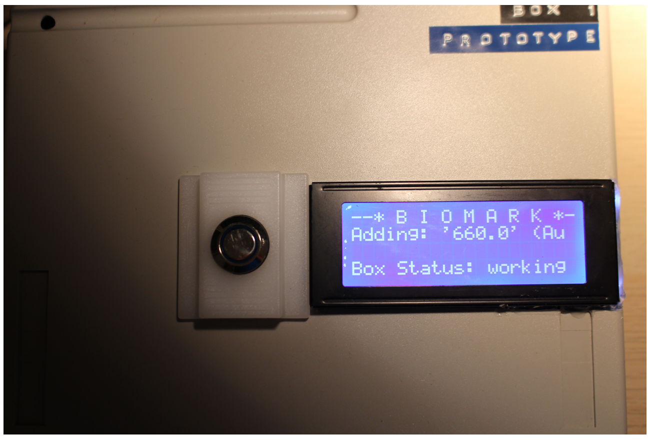


Figure 5.19: Calibration Box adding reagent automatically.

If there is not an automatic pump configured, the CB asks user to manually add the reagent and confirm the addition by pressing the CB button. The button LED blinks until the user presses it to continue with the next step of the calibration (figure 5.20).

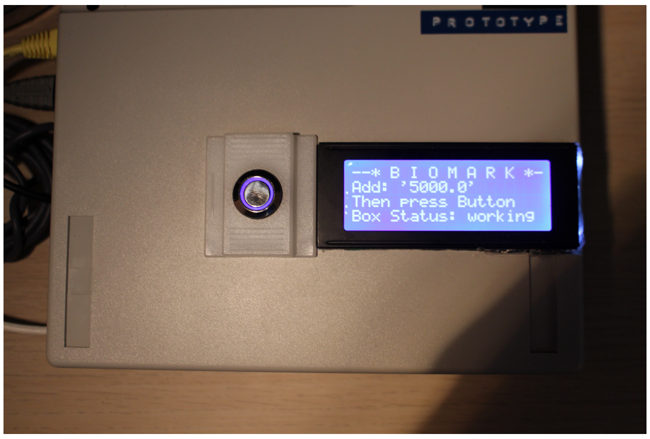
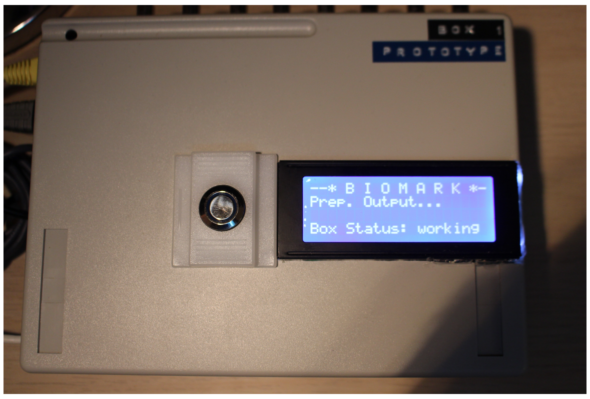
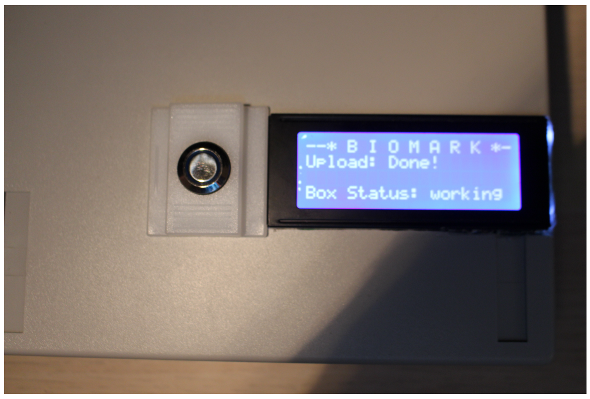


Figure 5.20: Calibration Box wating the addition of reagent manually.

This process, automatic or not, repeats the membrane testing against the continuous addition of concentration until the process is finished. Then the CB terminates calibration by processing results (figure 5.21, a) and uploading them as an excel file to the server (figure 5.21, b).



(a) Preparing data output.



(b) Uploading data.

Figure 5.21: Calibration Box (CB) terminating calibration process.

After upload the CB shows "finished" before goes back to the "free" status to receive the next calibration job (figure 5.22).

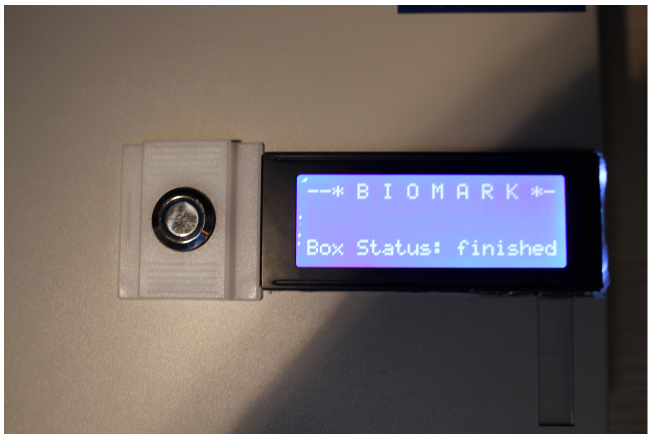
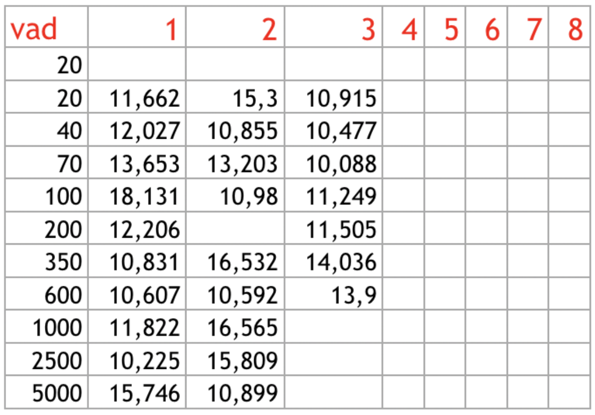


Figure 5.22: Calibration Box calibration terminated.

The automatic calibration output is an excel file uploaded to the server side application and available to download at the portal.

This excel file has the following format (figure 5.23).



vad	1	2	3	4	5	6	7	8
20								
20	11,662	15,3	10,915					
40	12,027	10,855	10,477					
70	13,653	13,203	10,088					
100	18,131	10,98	11,249					
200	12,206		11,505					
350	10,831	16,532	14,036					
600	10,607	10,592	13,9					
1000	11,822	16,565						
2500	10,225	15,809						
5000	15,746	10,899						

Figure 5.23: Calibration Box output calibration with three membranes.

This calibration is then processed by the researcher using excel or numbers (appendix B).

Chapter 6

Conclusion

In this chapter it is resumed all the work done during this project and described on the previous chapters. It will also be compared the proposal objectives and the achieved results. Finally it will be presented some future improvements or developments that can turn this project even better.

6.1 Achieved objectives

This project has as a main objective the research process automation regarding the membrane calibrations. Since the evolved equipments are not always available, this project had to be enough flexible, allowing the researcher to calibrate in a automatic or semi-automatic manner.

About automating the manual process, this goal was achieved by creating an automation interface called Calibration Box that interacts with the necessary equipments and with the researcher automating the full or part of the process and producing the same output that the researcher produces manually. To automate this process, there was a huge amount of detailed configurations and data entries to support the researcher on their calculations. This data was persisted and processed using PostgreSQL, Java (OpenJDK) and Spring Boot Frameworks.

The database relationships, and the need to clearly understood the calibration process to automate it, causes the detailed architecture definition prior the development of this solution, that was a great complement for this report.

The specific goals for this project was achieved and this prototype can be used to develop new units to automate the membrane calibration research process.

6.2 Future work

Still in the context of this work and in order to continue its evolution, it would be important to continue with the development process in two different ways.

Hardware Evolution - Develop a specific and more compact hardware module in a custom developed PCB that allow the reduction of hardware modules and assembly costs, making the Calibration Box more robust and compact.

Data Evolution - Part of the development process consists on the analysis of the collected values (data) and with this pattern analysis, the researcher advances with their research to produce a better and more reliable membrane to be used in a low cost way to easily detect cancer in early stages. This data is able to be processed and submitted to machine learning algorithms to suggest to the researcher new possibilities not only based on their calibrations, but based on all the researchers results.

6.3 Final considerations

In short, after evaluating all the aspects mentioned throughout the various chapters of this thesis, it is possible to distinguish this work for the ability to combine not only a software based solution, but also the ability to create a hardware based automation, distributing and isolating the data processing in the calibration process and contributing to a central interface allowing the researchers a faster way to calibrate their developments. In this project we can highlight the use of open-source hardware (Raspberry Pi) and software, reducing the prototype development costs, and the cloud based deployment using automated continuous integration and deployment. Throughout the various chapters were also presented all the steps from, architecture to the development of an hardware prototype and its software solution, and also the server-side solution enabling not only this automation but also the laboratory process organization.. This reports can act also as a user manual explaining how to use and automate a calibration.

Bibliography

[1] Y. Wang, Z. Zhang, V. Jain, J. Yi, S. Mueller, J. Sokolov, Z. Liu, K. Levon, B. Rigas, and M. H. Rafailovich, "Potentiometric sensors based on surface molecular imprinting: Detection of cancer biomarkers and viruses," *Sensors and Actuators B: Chemical*, vol. 146, no. 1, pp. 381 – 387, 2010.

[2] Y. Yin, Y. Cao, Y. Xu, and G. Li, "Colorimetric immunoassay for detection of tumor markers," *International Journal of Molecular Sciences*, vol. 11, pp. 5077–5094, Dec 2010.

[3] S. Ramos, "Cancer chemoprevention and chemotherapy: Dietary polyphenols and signalling pathways," *Molecular Nutrition & Food Research*, vol. 52, no. 5, pp. 507–526, 2008.

[4] G. M. Cooper, *The cell : a molecular approach / Geoffrey M. Cooper, Robert E. Hausman.* Sunderland, Mass. :: Sinauer Associates,, 2007.

[5] C. E. DeSantis, C. C. Lin, A. B. Mariotto, R. L. Siegel, K. D. Stein, J. L. Kramer, R. Alteri, A. S. Robbins, and A. Jemal, "Cancer treatment and survivorship statistics, 2014," *CA: A Cancer Journal for Clinicians*, vol. 64, no. 4, pp. 252–271, 2014.

[6] S. Sharma, "Tumor markers in clinical practice: General principles and guidelines," *Indian Journal of Medical and Paediatric Oncology*, vol. 30, no. 1, p. 1, 2009.

[7] N. L. Henry and D. F. Hayes, "Cancer biomarkers," *Molecular Oncology*, vol. 6, no. 2, pp. 140–146, 2012.

[8] M. A. Virji, D. W. Mercer, and R. B. Herberman, "Tumor markers in cancer diagnosis and prognosis," *CA: A Cancer Journal for Clinicians*, vol. 38, no. 2, pp. 104–126, 1988.

[9] R. Mayeux, "Biomarkers: Potential uses and limitations," *NeuroRX*, vol. 1, pp. 182–188, Apr 2004.

[10] K. Vytras, "The use of ion-selective electrodes in the determination of drug substances," *Journal of Pharmaceutical and Biomedical Analysis*, vol. 7, no. 7, pp. 789 – 812, 1989.

[11] R. Yan, S. Qiu, L. Tong, and Y. Qian, "Review of progresses on clinical applications of ion selective electrodes for electrolytic ion tests: from conventional ises to graphene-based ises," *Chemical Speciation & Bioavailability*, vol. 28, pp. 72–77, Apr 2016.

[12] A. R. Fakhari, M. Alaghemand, and M. Shamsipur, "Iron(iii)-selective membrane potentiometric sensor based on 5,10,15,20-tetrakis-(pentafluorophenyl)-21h,23h-porphyrin," *Analytical Letters*, vol. 34, pp. 1097–1106, Apr 2001.

[13] E. Pretsch, "The new wave of ion-selective electrodes," *TrAC Trends in Analytical Chemistry*, vol. 26, no. 1, pp. 46 – 51, 2007.

[14] A. H. KAMEL, S. A. A. ALMEIDA, M. G. F. SALES, and F. T. C. MOREIRA, "Sulfadiazine-potentiometric sensors for flow and batch determinations of sulfadiazine in drugs and biological fluids," *Analytical Sciences*, vol. 25, no. 3, pp. 365–371, 2009.

[15] I. O. for Standardization, *Quality Management Principles.* ISO, Geneva, Switzerland, 2015. ISBN: 978-92-67-10650-2.

[16] OpenJDK, "Openjdk 8." `https://openjdk.java.net/projects/jdk8/`.

[17] A. Maven, "Apache maven." `https://maven.apache.org/`.

[18] A. Foundation, "Apache license, version 2.0." `https://www.apache.org/licenses/LICENSE-2.0`.

[19] E. Foundation, "Eclipse public license - v 1.0." `https://www.eclipse.org/legal/epl-v10.html`.

[20] Oracle, "Common development and distribution license (cddl) version 1.1." `https://javaee.github.io/jstl-api/LICENSE`.

[21] BSD, "The 2-clause bsd license." `https://opensource.org/licenses/BSD-2-Clause`.

[22] M. License, "Mit license." `https://github.com/timcreative/freebies/blob/master/LICENSE.md`.

[23] P. Software, "Spring mvc." `https://docs.spring.io/spring/docs/current/spring-framework-reference/web.html`.

[24] Apache, "Apache tiles." `http://tiles.apache.org/index.html`.

[25] C. Tim, “Creative tim paper dashboard bootstrap template.” https://www.creative-tim.com/product/paper-dashboard.

[26] PostgreSQL, “About postgresql.” https://www.postgresql.org/about/.

[27] P. License, “Postgresql license.” https://www.postgresql.org/about/licence/.

[28] Docker, “What is docker, what is a container?.” https://www.docker.com/resources/what-container.

[29] P. Software, “About spring security.” https://spring.io/projects/spring-securityhttps://spring.io/projects/spring-security.

[30] B. team and contributors, “Bootstrap framework.” https://getbootstrap.com/.

[31] J. team and contributors, “Javamelody framework.” https://github.com/javamelody/javamelody.

[32] S. Software, “Swagger openapi specification.” https://swagger.io/resources/open-api/.

[33] D. Foundation, “Debian operating system webpage.” https://www.debian.org/.

[34] R. P. Foundation, “About raspberry pi.” https://www.raspberrypi.org/about/.

[35] R. P. Foundation, “Raspbian operating system webpage.” https://www.raspberrypi.org/downloads/raspbian/.

[36] Pi4J, “Pi4j library.” https://pi4j.com/1.2/index.html.

[37] G. Foundation, “Gnu general lesser public license (lgpl) version 3.0.” https://pi4j.com/1.2/license.html.

[38] Tinsharp, “Lcd with i2c hardware module.” https://www.makerguides.com/wp-content/uploads/2019/02/20x4-Character-LCD-Datasheet.pdf.

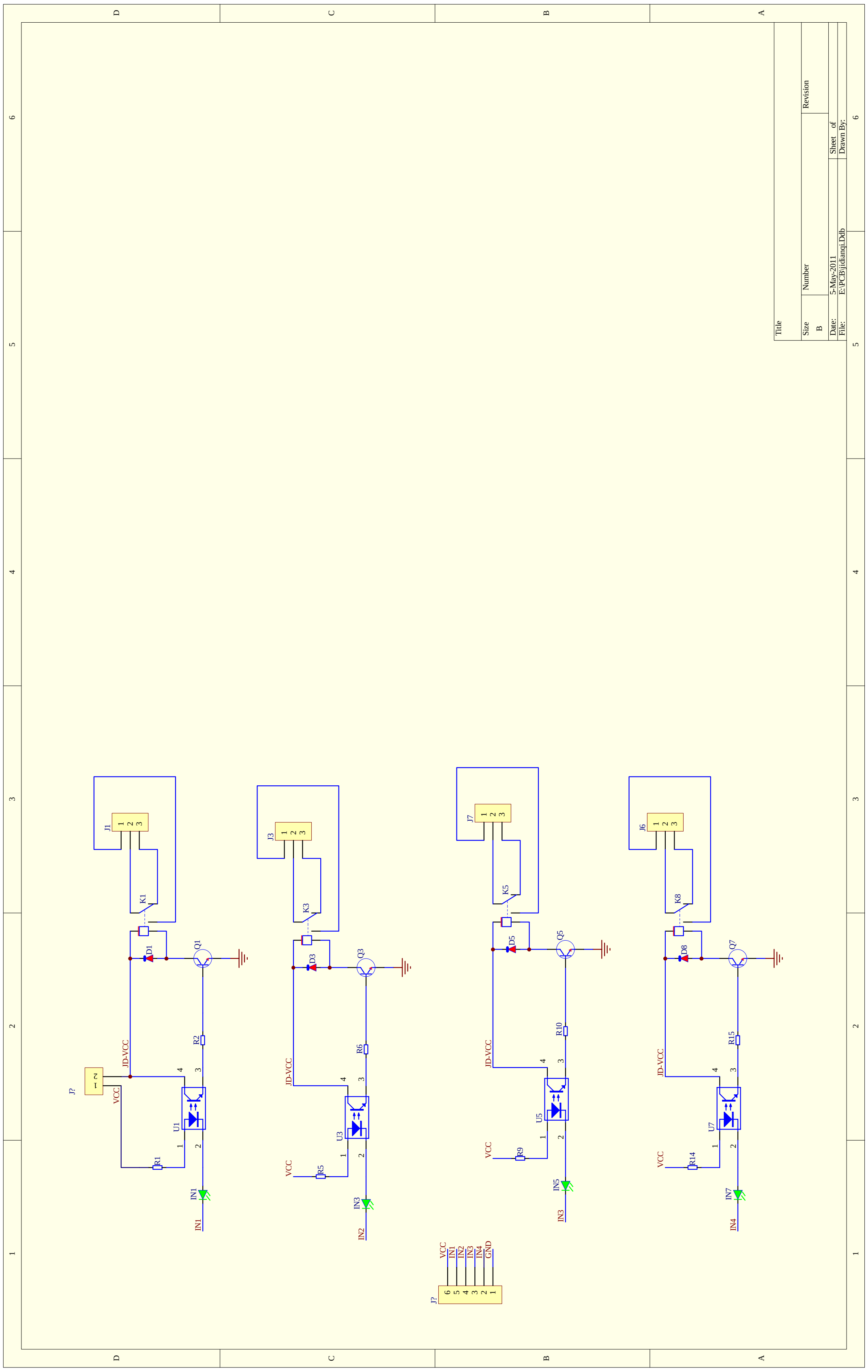
[39] Sunfounder, “Relay module diagram.” http://wiki.sunfounder.cc/images/1/1c/4_channel_relay_Schematic.pdf.

Appendix A

Hardware Modules

To build this project prototype, there were some external hardware modules included which his design is open-source, and could be integrated in a future work in a combined PCB [38], [39].

A.1 Relay Board Diagram

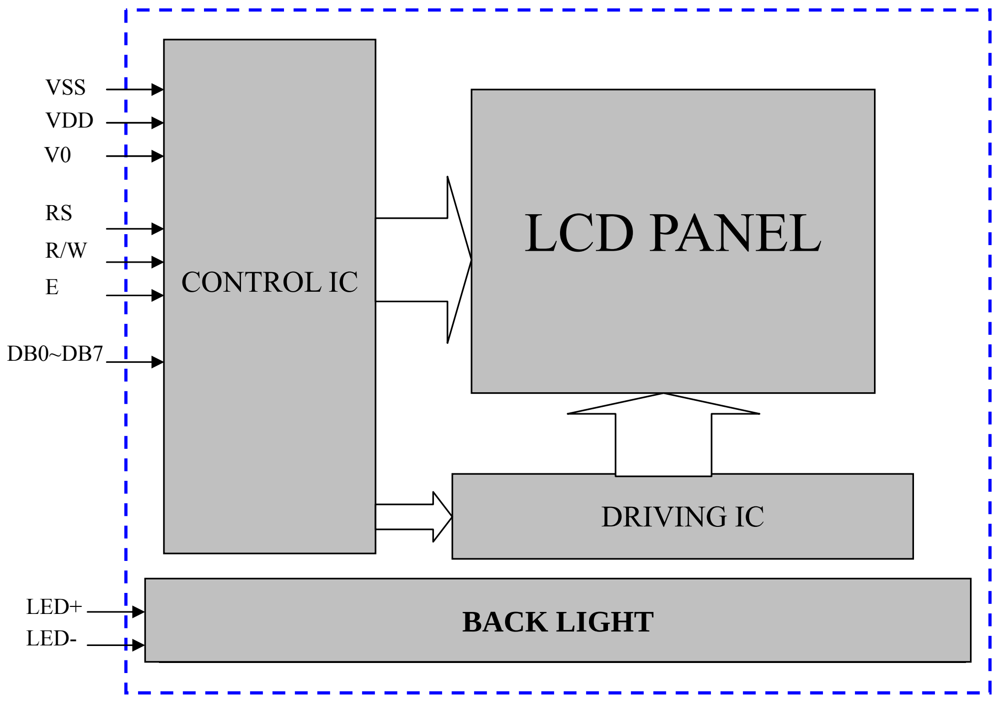


A.2 LCD Module Characteristics

FUNCTIONS & FEATURES

● Construction	: COB(Chip-on-Board)
● Display Format	: 20x4 Characters
● Display Type	: STN, Transflective, Positive, Y-G
● Controller	: SPLC780D1 or equivalent controller
● Interface	: 8-bit parallel interface
● Backlight	: yellow-green/ bottom light
● Viewing Direction	: 6 O'clock
● Driving Scheme	: 1/16 Duty Cycle, 1/5 Bias
● Power Supply Voltage	: 5.0 V
● V_{LCD} Adjustable For Best Contrast	: 4.7 V (V_{OP}.)
● Operation temperature	: -10℃ to +60℃
● Storage temperature	: -20℃ to +70℃

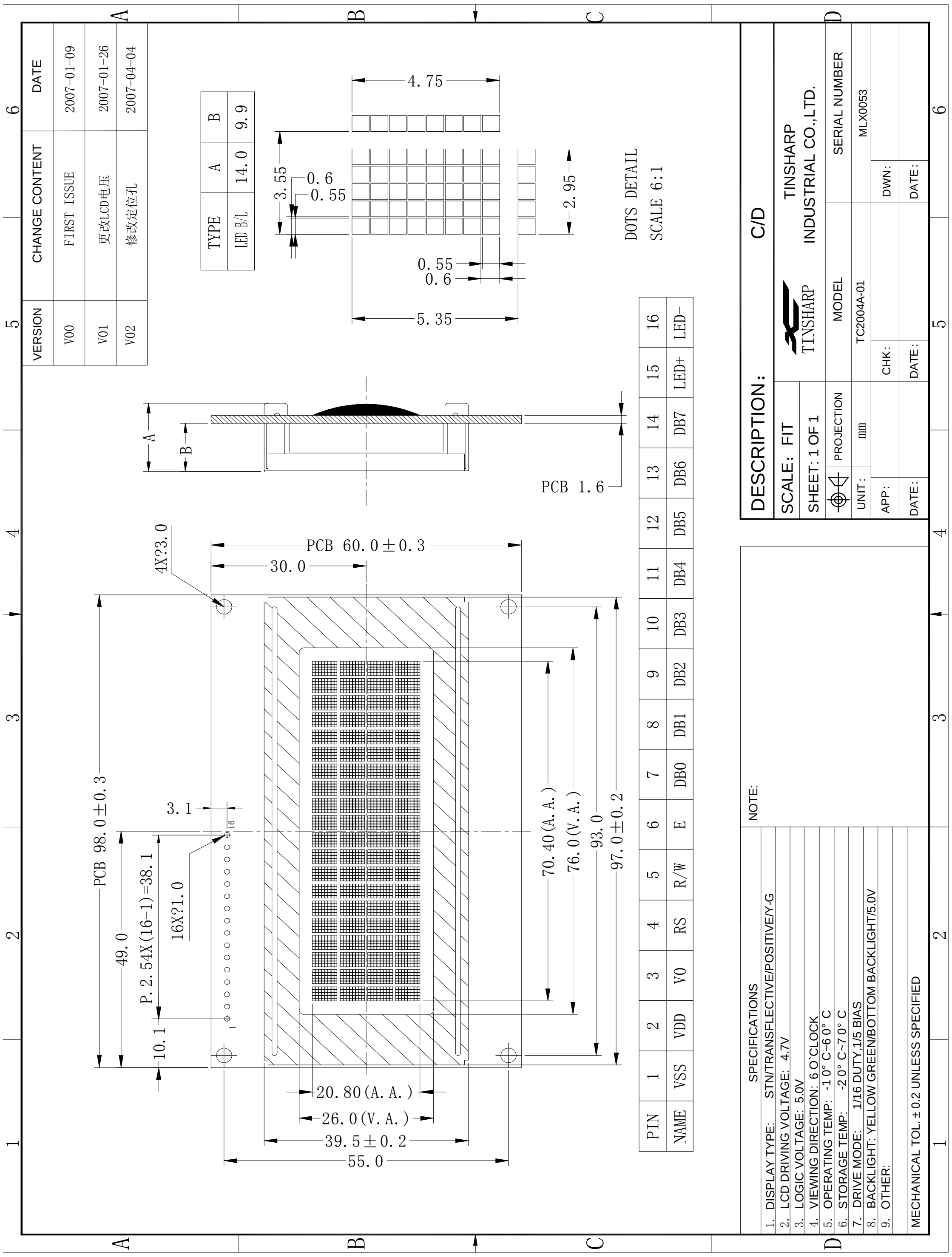
BLOCK DIAGRAM



A.3 LCD Module Characteristics

TC2004A-01

MODULE OUTLINE DRAWING



VERSION	CHANGE CONTENT	DATE
V00	FIRST ISSUE	2007-01-09
V01	更改LCD电压	2007-01-26
V02	修改定位孔	2007-04-04

TYPE	A	B
LED B/L	14.0	9.9

PIN	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
NAME	VSS	VDD	V0	RS	R/W	E	DB0	DB1	DB2	DB3	DB4	DB5	DB6	DB7	LED+	LED-

SPECIFICATIONS

1. DISPLAY TYPE: STN/TRANSFLECTIVE/POSITIVE/Y-G
2. LCD DRIVING VOLTAGE: 4.7V
3. LOGIC VOLTAGE: 5.0V
4. VIEWING DIRECTION: 6 O`CLOCK
5. OPERATING TEMP: -1 0° C~6 0° C
6. STORAGE TEMP: -2 0° C~7 0° C
7. DRIVE MODE: 1/16 DUTY,1/5 BIAS
8. BACKLIGHT: YELLOW GREEN/BOTTOM BACKLIGHT/5.0V
9. OTHER:

MECHANICAL TOL. ± 0.2 UNLESS SPECIFIED

NOTE:

DESCRIPTION: C/D

SCALE: FIT	TINSHARP	TINSHARP INDUSTRIAL CO.,LTD.	
SHEET: 1 OF 1			
PROJECTION	MODEL	SERIAL NUMBER	
UNIT: mm	TC2004A-01	MLX0053	
APP:	CHK:	DWN:	
DATE:	DATE:	DATE:	

Appendix B

Calibration Analysis

After collecting the membrane calibration data, the researcher starts its analysis using excel.

B.1 Processed Analysis

Calibração: **PIPES**			Static

Quantidade teórica

$C_4N_3H_7O$ Static

Creatinina	113,12	g/mol	Static
Conc.	1,0E-02	mol/L	Static

Sol.	0,5655895	g	Processed
Stock	0,500	L	Static

Quantidade pesada

Creatinina	113,12	g/mol	Static
Conc.	1,0E-02	mol/L	Processed

Sol.	0,05658	g	Static
Stock	0,050	L	Static

	pH=2,85	Static
Vinicial	**HEPES**	Static
50 mL		Static

Quantida de teórica

PIPES	302,37	g/mol	Static
Conc.	5E-03	mol/L	Static
Sol.	0,75593	g	Processed
Stock	0,500	L	Static

Quantidade pesada

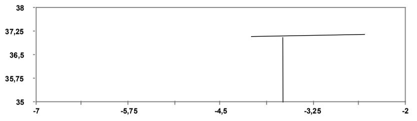
PIPES	302,37	g/mol	Static
Conc.	1E-03	mol/L	Processed
Sol.	0,75656	g	Static
Stock	0,500	L	Static

Legenda

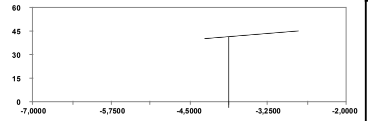
I	MIP 2
II	NIP 2

			mV sem sol. Carnitina	39,1	31,8	-1,5	-266,8	55,5	125,3	126,7	-266	-37,2	-235,4	155,4	224,5
Vad(mL)	**Vt ad(mL)**	**Conc, M**	**log C**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**3 (suplente)**
20	20	4,00E-06	-5,3980	40,1	32,2	-2,5	-264,0	55,5	125,1	126,9	-262,6	-36,8	-229,5	155,4	224,8
20	40	8,00E-06	-5,0971	39,8	32,2	-2,3	-260,0	55,7	125,1	126,4	-259,1	-34,5	-224,0	155,0	224,0
40	80	1,60E-05	-4,7964	39,7	32,7	-2,1	-253,7	55,7	125,4	125,9	-254,4	-36,5	-217,5	154,5	224,1
70	150	2,99E-05	-4,5240	39,6	32,5	-2,1	-243,7	56,1	125,8	125,8	-242,8	-36,2	-207,4	154,1	223,5
100	250	4,98E-05	-4,3030	39,4	32,5	-2,0	-233,2	56,0	125,3	125,3	-232,8	-35,3	-198,2	153,7	223,2
200	450	8,92E-05	-4,0495	38,0	31,2	-2,7	-220,6	54,5	125,0	125,0	-219,8	-35,8	-186,0	153,6	222,7
350	800	1,58E-04	-3,8026	36,6	29,7	-3,4	-207,1	52,8	125,0	125,0	-206,2	-36,0	-173,6	153,5	22,4
600	1400	2,72E-04	-3,5647	36,0	29,1	-2,5	-192,1	53,2	124,8	124,8	-193,4	-37,5	-161,2	153,6	222,8
1000	2400	4,58E-04	-3,3390	33,3	25,8	-2,7	-180,3	49,9	124,3	124,3	-181,4	-40,7	-149,3	153,0	221,9
2500	4900	8,93E-04	-3,0492	26,6	13,5	-5,6	-167,1	40,0	124,6	124,6	-166,0	-49,8	-133,6	153,2	222,2
5000	9900	1,65E-03	-2,7816	14,7	-19,0	-7,1	-154,5	13,4	123,5	123,5	-153,6	-78,3	-121,0	152,5	220,9
Static	Processed	Processe	Processed												

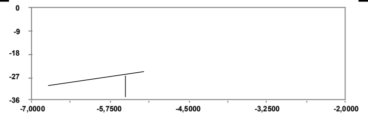
log C	1	1
-5,3980	40,1	40,1
-5,0971	39,8	39,8
-4,7964	39,7	39,7
-4,5240	39,6	39,6
-4,3030	39,4	39,4
-4,0495	38,0	38,0
-3,8026	36,6	36,6
-3,5647	36,0	36,0
-3,3390	33,3	33,3
-3,0492	26,6	26,6
-2,7816	14,7	14,7



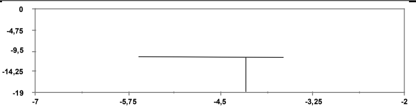
log C	5	5
-5,3980	55,5	55,5
-5,0971	55,7	55,7
-4,7964	55,7	55,7
-4,5240	56,1	56,1
-4,3030	56,0	56,0
-4,0495	54,5	54,5
-3,8026	52,8	52,8
-3,5647	53,2	53,2
-3,3390	49,9	49,9
-3,0492	40,0	40,0
-2,7816	13,4	13,4



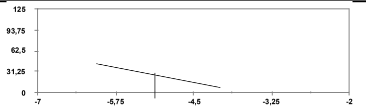
log C	9	9
-5,3980	-36,8	-36,8
-5,0971	-34,5	-34,5
-4,7964	-36,5	-36,5
-4,5240	-36,2	-36,2
-4,3030	-35,3	-35,3
-4,0495	-35,8	-35,8
-3,8026	-36,0	-36,0
-3,5647	-37,5	-37,5
-3,3390	-40,7	-40,7
-3,0492	-49,8	-49,8
-2,7816	-78,3	-78,3



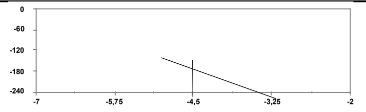
log C	2	2
-5,3980	32,2	32,2
-5,0971	32,2	32,2
-4,7964	32,7	32,7
-4,5240	32,5	32,5
-4,3030	32,5	32,5
-4,0495	31,2	31,2
-3,8026	29,7	29,7
-3,5647	29,1	29,1
-3,3390	25,8	25,8
-3,0492	13,5	13,5
-2,7816	-19,0	-19,0



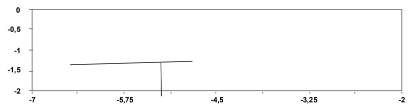
log C	6	6
-5,3980	125,1	125,1
-5,0971	125,1	125,1
-4,7964	125,4	125,4
-4,5240	125,8	125,8
-4,3030	125,3	125,3
-4,0495	125,0	125,0
-3,8026	125,0	125,0
-3,5647	124,8	124,8
-3,3390	124,3	124,3
-3,0492	124,6	124,6
-2,7816	123,5	123,5



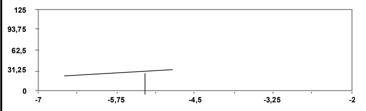
log C	10	10
-5,3980	-229,5	-229,5
-5,0971	-224,0	-224,0
-4,7964	-217,5	-217,5
-4,5240	-207,4	-207,4
-4,3030	-198,2	-198,2
-4,0495	-186,0	-186,0
-3,8026	-173,6	-173,6
-3,5647	-161,2	-161,2
-3,3390	-149,3	-149,3
-3,0492	-133,6	-133,6
-2,7816	-121,0	-121,0



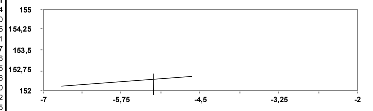
log C	3	3
-5,3980	-2,5	-2,5
-5,0971	-2,3	-2,3
-4,7964	-2,1	-2,1
-4,5240	-2,1	-2,1
-4,3030	-2,0	-2,0
-4,0495	-2,7	-2,7
-3,8026	-3,4	-3,4
-3,5647	-2,5	-2,5
-3,3390	-2,7	-2,7
-3,0492	-5,6	-5,6
-2,7816	-7,1	-7,1



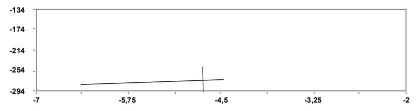
log C	7	7
-5,3980	126,9	126,9
-5,0971	126,4	126,4
-4,7964	125,9	125,9
-4,5240	125,8	125,8
-4,3030	125,3	125,3
-4,0495	125,0	125,0
-3,8026	125,0	125,0
-3,5647	124,8	124,8
-3,3390	124,3	124,3
-3,0492	124,6	124,6
-2,7816	123,5	123,5



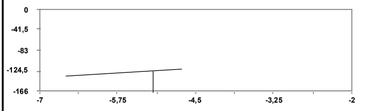
log C	11	11
-5,3980	155,4	155,4
-5,0971	155,0	155,0
-4,7964	154,5	154,5
-4,5240	154,1	154,1
-4,3030	153,7	153,7
-4,0495	153,6	153,6
-3,8026	153,5	153,5
-3,5647	153,6	153,6
-3,3390	153,0	153,0
-3,0492	153,2	153,2
-2,7816	152,5	152,5



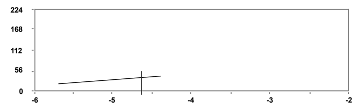
log C	4	4
-5,3980	-264,0	-264,0
-5,0971	-260,0	-260,0
-4,7964	-253,7	-253,7
-4,5240	-243,7	-243,7
-4,3030	-233,2	-233,2
-4,0495	-220,6	-220,6
-3,8026	-207,1	-207,1
-3,5647	-192,1	-192,1
-3,3390	-180,3	-180,3
-3,0492	-167,1	-167,1
-2,7816	-154,5	-154,5



log C	8	8
-5,3980	-262,6	-262,6
-5,0971	-259,1	-259,1
-4,7964	-254,4	-254,4
-4,5240	-242,8	-242,8
-4,3030	-232,8	-232,8
-4,0495	-219,8	-219,8
-3,8026	-206,2	-206,2
-3,5647	-193,4	-193,4
-3,3390	-181,4	-181,4
-3,0492	-166,0	-166,0
-2,7816	-153,6	-153,6



log C	3 (suplente)	3 (suplente)
-5,3980	224,8	224,8
-5,0971	224,0	224,0
-4,7964	224,1	224,1
-4,5240	223,5	223,5
-4,3030	223,2	223,2
-4,0495	222,7	222,7
-3,8026	22,4	22,4
-3,5647	222,8	222,8
-3,3390	221,9	221,9
-3,0492	222,2	222,2
-2,7816	220,9	220,9



ESI	Slope (mV/decade)	R^2 (n = 5)	LOD (M)	LLLR (M)
I				
II				
III				
IV				
I				
II				
III				
IV				
I				
II				
III				
IV				

1