Universidade de Aveiro, Departamento de Electrónica, Telecomunicações e Informática, 2019

VIRHUS

Digital laboratory for the simulation of human physiology and behavior

Supervisors:
Ilídio Oliveira
Susana Brás

Autores/ Authors:
Gonçalo Ferreira, Nº Mec 68761
Francisco Oliveira, Nº Mec 77091
Isaac dos Anjos, Nº Mec 78191

VIRHUS

Digital laboratory for the simulation of human physiology and behavior

Digital laboratory for the simulation of human physiology and behavior Informatic project, integrated with in Informatic Engineering Bachelor Degree, conducted at the University of Aveiro performed by Gonçalo Ferreira, Isaac dos Anjos and Francisco Oliveira under the guidance of the Professor Ilidio Oliveira and the researcher Susana Brás.

Acknowledgements

We would like to thank our supervisors, Professor Ilídio Oliveira and Researcher Susana Brás for their support during the execution of the project

Table of Content

Table of Figures

Introduction

VIRHUS, short for Virtual Humans, is a perfect example of how all innovative projects started off, converting mechanical solutions into software projects.This method of digitalising analogical devices into online service might sound trivial to do but many existing transitions required more than just programming.

VIRHUS is a multi platform designed to simulate clinical signals through predefined conditions. Breaking it down, having “virtual humans” stimulate a certain emotion while returning data about heart rate, muscle movement and sweat measurement. This data gives developers and researchers a foundation to their studies

Motivation

Data science has started to trend over the decade, and throughout we’ve seen platforms emerging in this area giving support and data to research development. The motivation behind this is to contribute revellante tools to data projects specialized in medical analysis.
For researchers to conduct medical studies on human organs they need access to data. Popular websites like Physionet are limited and usually contains one deviation of data and in undesirable sizes.

Goal

The acquisition of physiological and behavioral signals of the human being is not trivial, because we need participants and this ends up not practical, as much for economic as well as ethical questions.

The goal of this project is to create clinical signals under the effect of emotions and substances, develope and intuitive online platform and a peripheral mobile application for offline purposes. These clinical signals are produced through the platform, where the user can adjust the emotions through the live experiment.

Diving the project into three main parts, first one being the signal generator, given preconditions, length and triggering various emotions, the generator would return the requested signal.

To assure that the system can produce 2 hours of data, the system should generate only when the amount of data in queue low, thus reducing huge payloads of data and intensive processing. The second part targets the web application where users can interact with the system, targeting developers and researchers. The goal here is to create, play export and delete data. The playable feature should not consume the user’s at any point.

Document structure

The report is organized as follows:

Chapter one - Clinical signals: A brief explanation of some types of signals that we use in our project.

Chapter two - State of the Art: Some of the work developed in this area is presented, namely similar systems or that somehow can be compared to the one developed by us. It also presents justifications for certain technological choices made by the team in the development of VIRHUS.

Chapter three - System Requirements and Architecture: Exposure of all requirements analysis and system architecture.

Chapter four - Implementation: Description of how the solution presented by us was implemented, detailing each of the modules on which the system is based: Acquisition, data processing and visualization/ consumers.

Chapter five- Conclusion and Future work: Conclusion of the document, presenting final considerations about the developed system, and indications about everything that could be done in the future to complete the presented solution. The web application should also display data related to each virtual human. The communication used should be check if the
If the connection was made by an authenticated user through sessions or tokens.

Chapter 1 Clinical signals

1. Clinical Signals

1.1 What are clinical signals?

Clinical signals are readings made by highly sophisticated machines conducted on patients. These machines are precisely designed to read/interpret the status of a specific organ of the body, like heart, skin, muscles etc. The data of each organ has a technical name associated:

- Heart rate: ECG
- Electrodermal activity on the skin: EDA
- Electrical activity produced by the skeleton muscles: EMG
- Nerve Conduction Velocity: NCV
- ELectrical signals present in the brain/ spinal cord: SSEP

1.2 What's the context of these signals within the project?

Information systems, in the past, relied on real data(data from patients) or from a generator machine to properly grow, grow in the sense that the testing phase conducted realistic scenarios. The list of signals referred above are the signals the system will support.

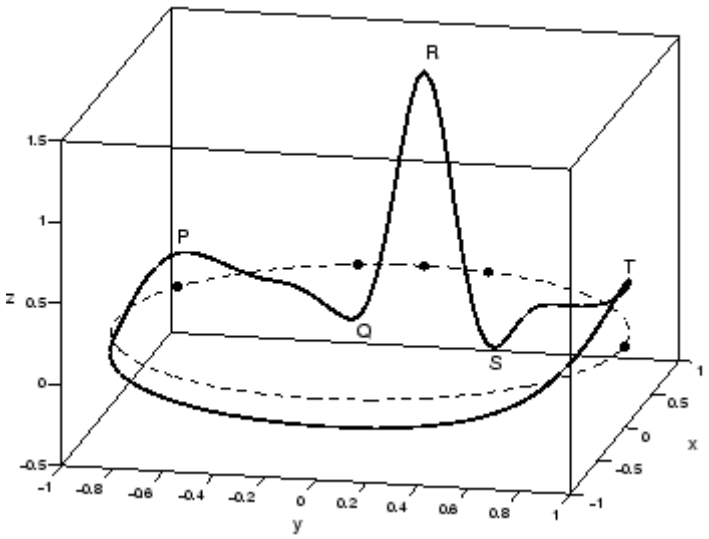
2. State of the Art

2.1 Related Projects

The objective of this section is to provide projects and studies whos goals intersected with this project. Many studies were conducted to create methods of testing and maintenance of medical equipment, and many of them used patient data as a starting point. A quick insight of these studies illustrated below.

2.1.1 Physionet A Realistic ECG Waveform Generator ECGSYN

ECGSYN generates a synthesized ECG signal using a model based on three coupled ordinary differential equations. The software simulates many of the features of the human ECG, including beat-to-beat variation in morphology and timing, respiratory sinus and R-peak amplitude modulation. The output of ECGSYN may be employed to properly assess biomedical signal processing techniques which are used to compute statistics from the ECG



1 ECGSYN WAVEFORM

2.1.2 Ambulance in a Box

(Development and Implementation of a low budget simulation center for clinical emergencies) By Ramiro Pozzo and Alfred Pacheco, the project is about a desktop application, designed to simulate physiologic data and generate the presentation of the patient's vital signs and adapt the backend of an unused ambulance.

The argentino duo used Pascal and Delphi programming tools to provide the operator various clinical signals and shapes according to the real vital sign information and display formats. Since the clinical signals can vary in trauma vital signs, the authors decided to use realistic trauma and shock patients during the ambulance transport. Why was not explained but considering the high variance of this topic and budget wise, was probably the correct decision. Also important how the authors represented the data and related info.

2.1.3 Biomedical Engineering Systems and Technology by Mauricio Tavares and Tiago Rockerbach.

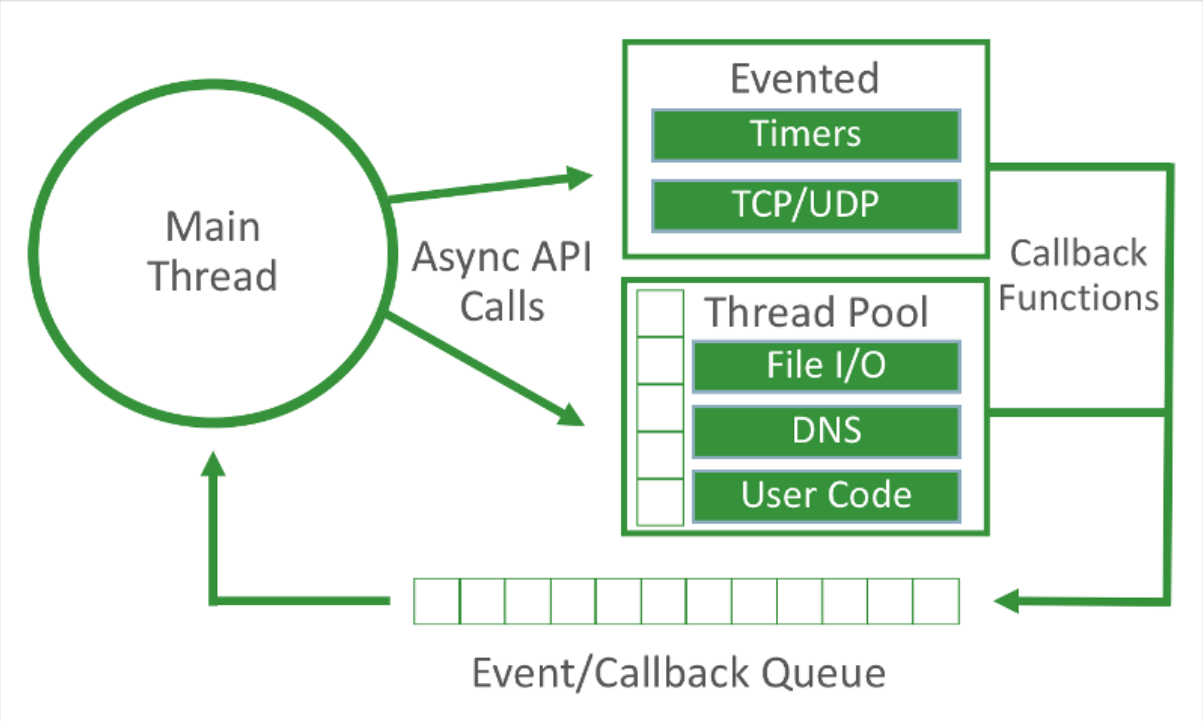
This paper illustrates a micro converter based electronic device developed to simulate auditory evoked potentials of ECG and electronystagmography signals. The reason behind this study is to periodically assess medical equipment since patients maybe harmed if they do not work properly. The authors used sampled AEP and ECG waveforms over mathematical equations. The challenges in this in this paper are replacing performance analysers, due to their expense and testing the performance of medical equipment.
In addition, properly designing a PCB payout and carefully programming the ADU micro converters. The author discusses how to test medical equipment through the simulator, explaining the standards of each signal and the wellbeing of the patient.

2.2 Technology

In this section exhibits the technology used throughout the project, presenting a brief description, pros and cons, reason of choice a main features.

2.2.1 NodeJS

Nodejs is an open-source, Javascript runtime environment that executes Javascript code outside of a browser. This allows developers to write command line tools and for server-side scripting to produce dynamic web page content before displaying the page to the user's web browser. NOdeJS uses V8 engine which was originally developed for the Chrome browser, used to compile functions written in Javascript into a machine code at an impressive speed.



2 Asynchronous Diagram

Nodejs was designed heavily as an asynchronous event driven JavaScript runtime, making it possible to build scalable network applications. In the example above represents how asynchronous functions behave in Nodejs. Every program, server or application starts off on the main thread, thread responsible for all sub threads and terminating the program. Within the main thread, imagine the program needs to load configurations from a file, perform a couple of http requests and so do some calculations. Instead of performing a linear execution, NodeJS can run all 3 of these tasks on separate threads and have them return through a callback function, which are functions called after the process return or terminates.

Pros Using NodeJS for backend development:

. Robust technology stack, how the language Javascript is shared in various components in a project. This brings better efficiency and overall developer productivity, code sharing, reusability and knowledge sharing with a team.

. Fast processing and event-based model; NodeJS is fast, handling concurrent requests much quicker than GO, PHP and Java does. Non-blocking input/output and asynchronous request handling made NodeJS capable at processing requests without any delay. Another aspect is the event-based model. In a client/server side scenario, synchronization happens fast, good for real-team applications. In NodeJS, due to its asynchronous non-blocking single-thread nature, it is the most popular choice for online gaming, chats, video conferences or any solution requiring updated data.

. Scalable Technology for microservices; NodeJS is also a great choice for microservices, breaking an application logic into modules, microservices, instead into one service.

. Strong corporate support; Many companies like IBM, Microsoft, Paypal helped accelerate the development of NodeJS giving hundreds of other companies reliable alternative to PHP and Java.

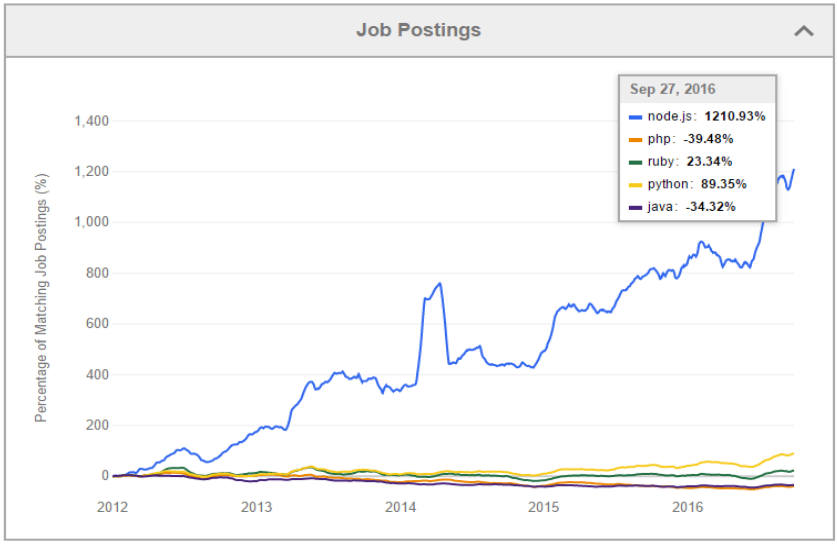
Cons Drawbacks of using NodeJS

. Performance bottlenecks with heavy computation tasks; the inability to process CPU bound tasks;

. Callback hell issue; Nesting callback functions inside callbacks becomes difficult to understand and maintain the code.

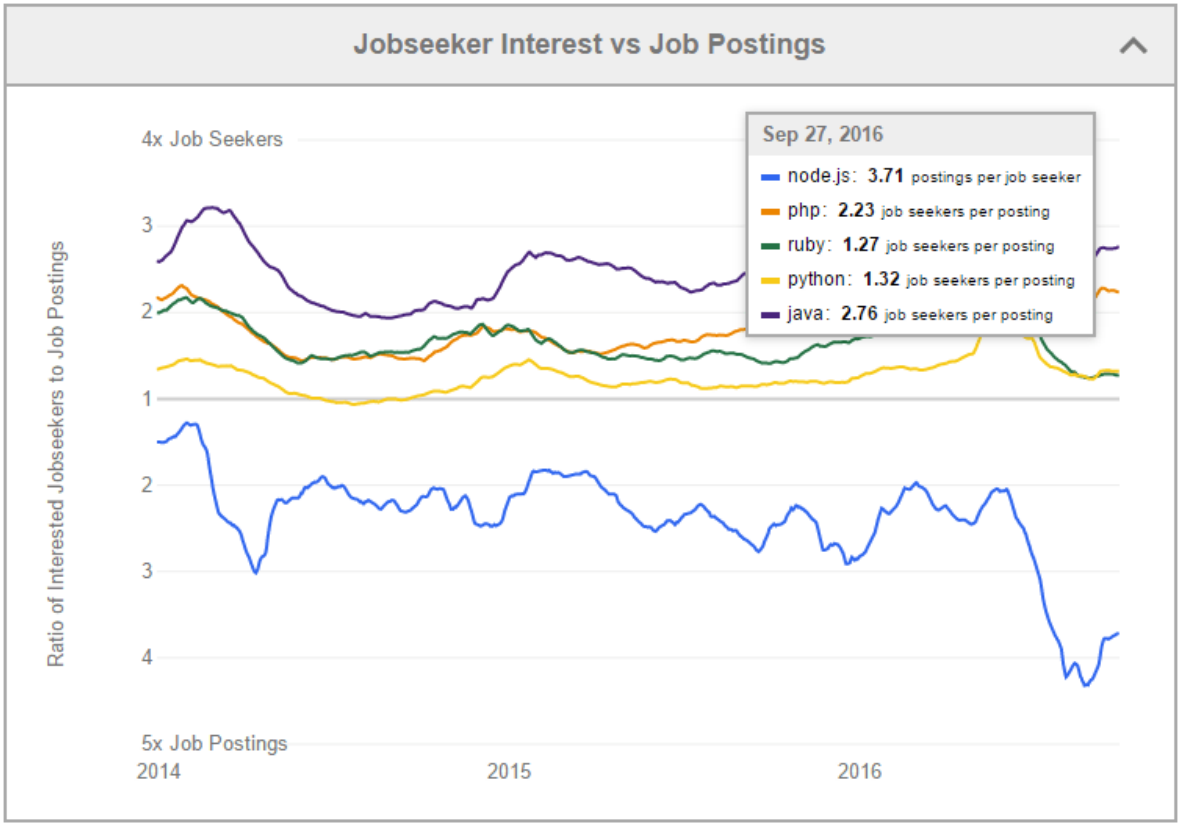
. Immaturity of tooling; Node Package Manager(NPM) registry holds either poor quality or not properly documented/tested tools.

Reason in why we chose NodeJS
As the interest over Node.js continues to grow, the demand for experienced professionals in this field also increases.



3 Node.js Job posting

Being relatively new, it's still far from meeting this demand of professionals. According to data from Indeed.com, the number of Node.js job postings is 3-4 times bigger than the number of job seekers with a similar skillset.



4 Node.js Job Seeker

From an investing standpoint, adding time and effort into working with Node.js is a must do, especially in a learning environment like College or professional courses.

Another reason for including Node.js into this project was due to how many modules/tools existed in the remote registry, NPM, and documentation available to each one.
Modules used in our project:

- Express; minimal and flexible Node.js web application framework.
- Express-sessions; handling login sessions
- Http; Built in http server for handling Http requests
- Socket.io; for websocket communications
- Amqplib; RabbitMQ Client

2.3.2 Rickshaw.js

Rickshaw is a Javascript toolkit for creating interactive time series graphs, developed by Shutterstock. Rickshaw also offers

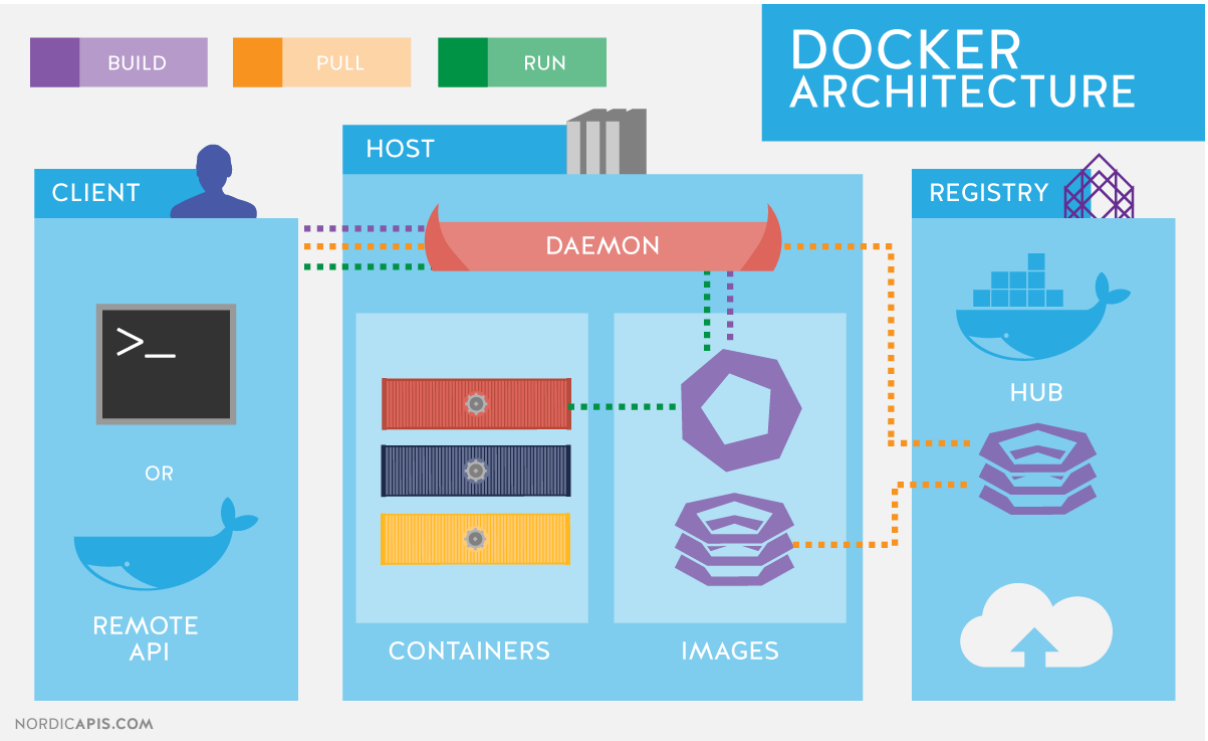
- Lines and Toggling
- Interactive Real-Time Data
- Scatter Plot with Multiple Series
- Stacked Bars with Deterministic Colors
- Color Schemes
- Data via AJAX
- Y-Axis tick marks
- Custom Values
- Fixed Window Series
- Hover Details Custom Formatting
- Scaled Series
- Multiple Renderers

2.3.3 PostgreSQL

PostgreSQL is an open source object-relational database system that uses and extends the SQL language combining with other features to safely store and scale the most complicated data.
PostgreSQL comes with many features aimed to help developers build applications, administrators to protect data integrity and build fault-tolerant environments, and help you manage your data no matter how big or small the dataset. In addition to being free and open source, PostgreSQL is highly extensible. For example, you can define your own data types, build out custom functions, even write code from different programming languages without recompiling your database!

2.3.4 Docker

Docker is a set of coupled software and platform as a service products that use operating system level virtualization to develop and deliver software in packages called containers. The containers are hosted by the Docker Engine



5 Docker Diagram

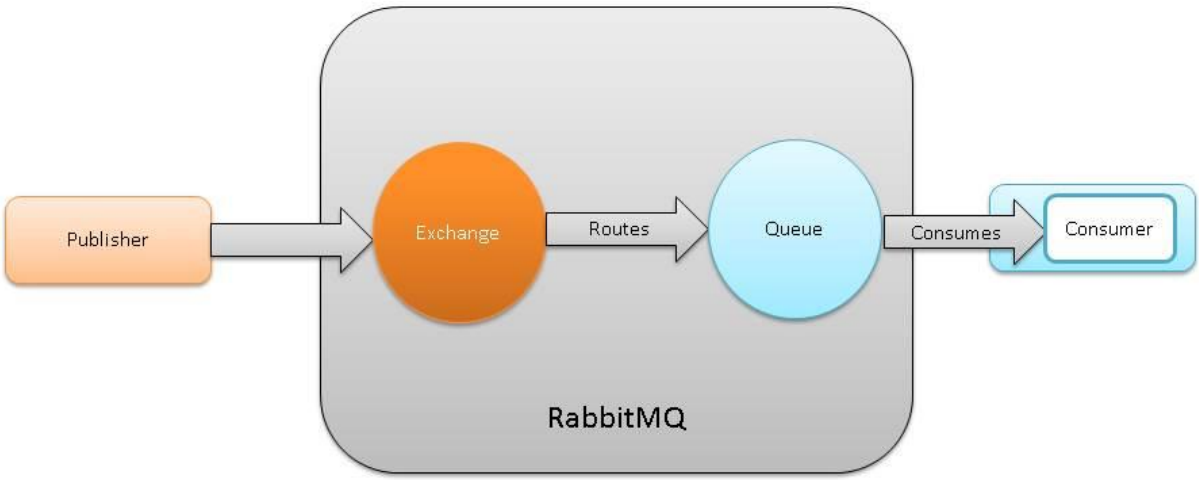
A container is a standard unit of software that packages up code and all its dependencies so applications can run from one computing environment to another. Where as a container image is a lightweight executable package of software that includes everything needed to run an application. Containers isolate software from its environment and ensure that it works uniformly regardless of the development stage.

2.3.5 RabbitMQ

RabbitMQ is an open-source message-broker software, lightweight and easy to deploy on premises. Supports multiple messaging protocols and be deployed in distributed and federated configurations to meet requirements.
RabbitMQ offers features like:

- Asynchronous Messaging
- Distributed Deployment
- Management & Monitoring
- Tools & Plugins

RabbitMQ runs on many operating systems and cloud environments and providing a wide rage of developer tools for most popular languages.



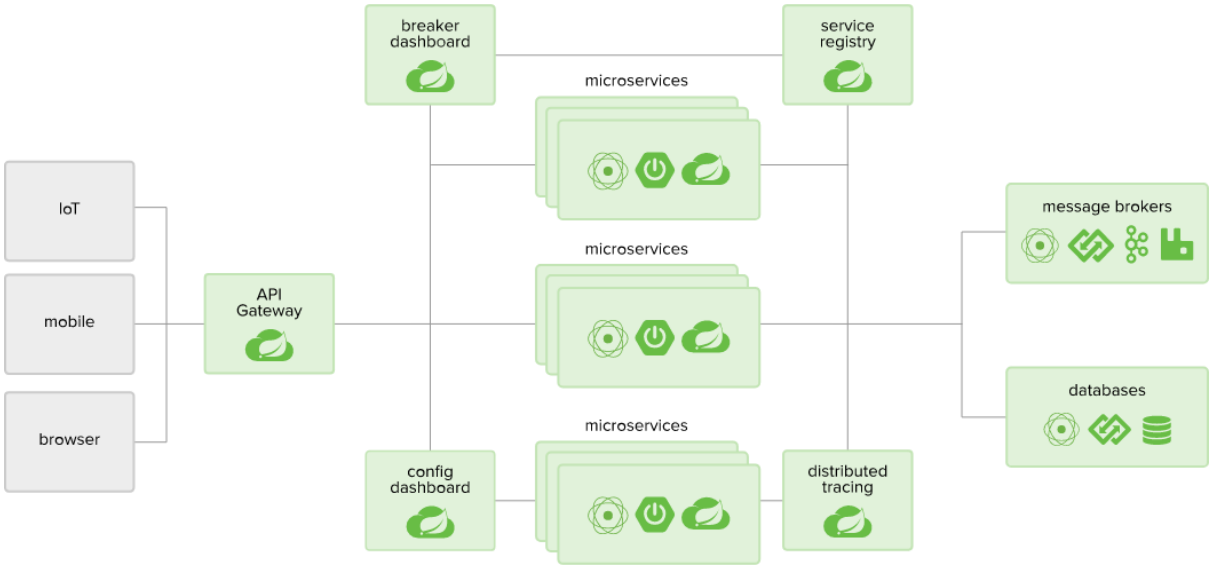
6 RabbitMQ Diagram

A message broker is an architectural pattern for message validation and routing. It resolves communication among applications, removing the mutual awareness that application should have of each other in order to be able to exchange messages effectively.

The purpose of a broker is to receive incoming messages from applications and perform any routing/ modification on them. Thus decoupling end-points, which facilitate reuse of intermediary functions.
Brokers are based on one of two fundamental architectures hub-and-spoke and message bus. Hub-and-spoke,the server acts as the mechanism that provides integration services, where as the message bus is a communication backbone or distributed service.

2.3.6 Spring Boot RESTful API

Spring Boot is an open source Java-based framework used to create a micro Service. Spring Boot provides a good platform for Java developers to develop a stand-alone and production-grade spring application that you can just run. It is easy to understand and develop spring applications, increases productivity and reduces development time. Provides a flexible way to configure Java Beans, XML configurations, and Database Transactions. Everything is auto configured, no manual configurations are needed. It offers annotation-based spring application, eases dependency management and includes Embedded Servlet Container.
A REST API defines a set of functions which developers can perform requests and receive responses via HTTP protocol such as GET and POST. It's a requirement of a REST API that the client and server are independent of each other to allow improvements along the time without interfering with each other.



7 Springboot Diagram

2.3.7 Android Studio

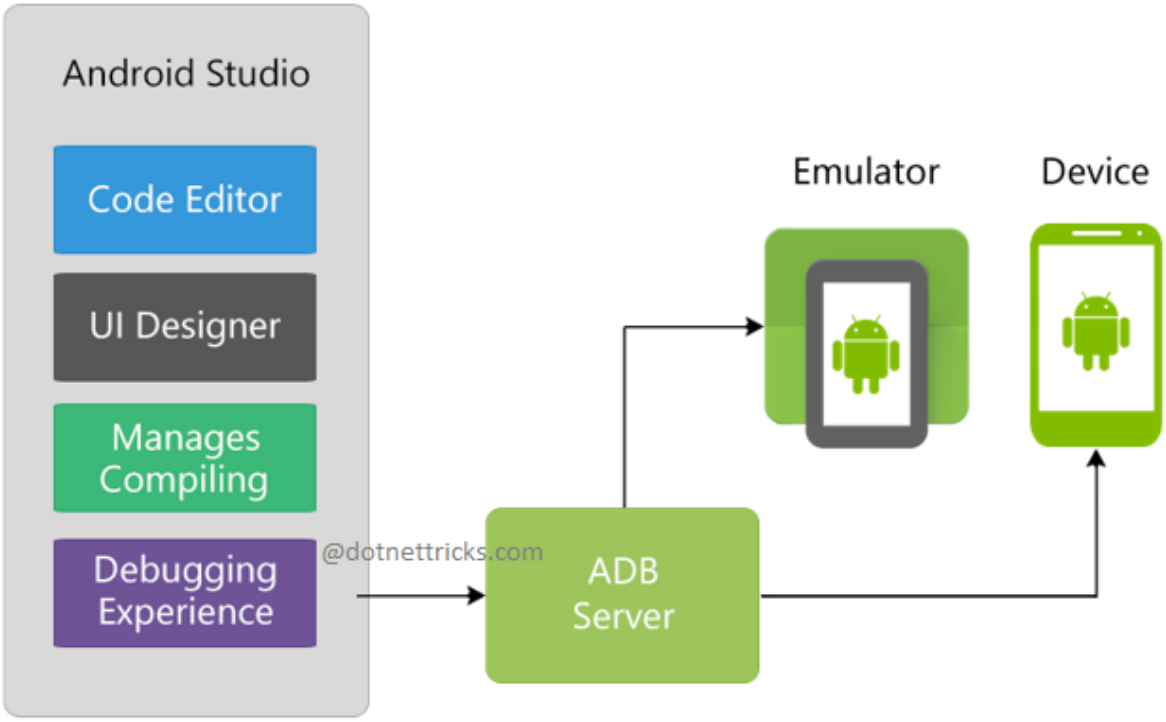
Android Studio is an official Integrated Development Environment (IDE) designed to create Android applications based on IntelliJ IDEA.
It offers even more features that enhance your productivity when building Android apps, such as:

- A flexible Gradle-based build system
- A fast and feature-rich emulator
- A unified environment where you can develop for all Android devices
- Instant Run to push changes to your running app without building a new APK
- Code templates and GitHub integration to help you build common app features and import sample code

- Extensive testing tools and frameworks
- Lint tools to catch performance, usability, version compatibility, and other problems
- C++ and NDK support
- Built-in support for Google Cloud Platform, making it easy to integrate Google Cloud Messaging and App Engine

Each app module contains the following folders:

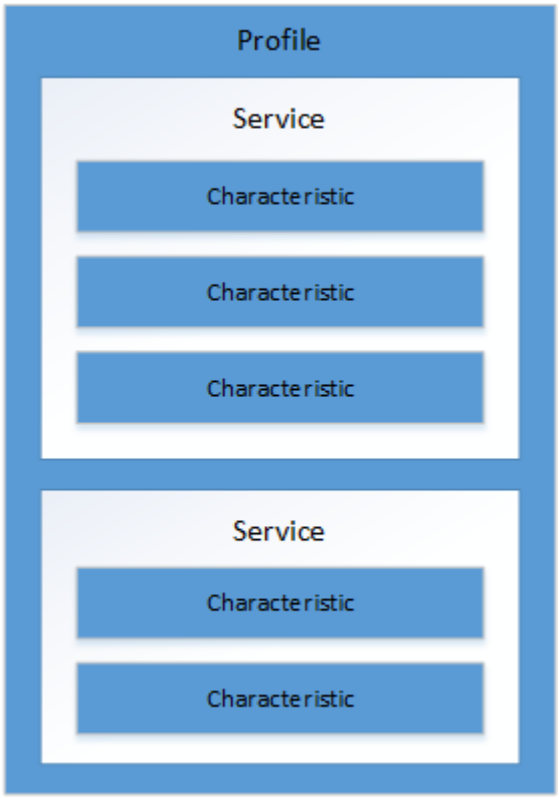
- manifests - Contains the AndroidManifest.xml file.
- java - Contains the Java source code files, including JUnit test code.
- res - Contains all non-code resources, such as XML layouts, UI strings, and bitmap images.



8 Android Studio Diagram

2.3.8 Bluetooth Low Energy Technology

Bluetooth Low Energy is a wireless personal area network technology designed by the Bluetooth Special Interest Group aimed to support applications in the healthcare, fitness and home entertainment industries. Manufacturers are expected to implement the appropriate specifications for their device in order to ensure compatibility. Specifications are in the BLE domain are known as Generic Attribute Profile(GATT), a standard for sending and receiving short pieces of data. GATTs are profiles that contains services and within each service are characteristics.



9 BLE Diagram

On Bluetooth.com, Bluetooth SIG team has gathered a list of existing profile specifications. Many to which might be useful for this project:

- Heart Rate Profile
- Health Thermometer Profile
- Blood Pressure Profile

2.3 Conclusions

After reviewing these projects, there are interesting viewpoints that could be used for this project. In the paper Ambulance in a box by Ramiro Pozzo, the author discussed about generating trauma patient vital signs. The data used were clinical signals of real trauma and shock patients during the ambulance transport. Also consider information like heartbeats per minute,ECG waveform shapes important aspects when displaying data to the user.

On the other hand, ECGSYN software contributed to Physionet by Patrick McSharry shows a different method of generating data. In this project, the author used a mathematical model to create realistic ECG signals.

Comparing both methods, the first one relies highly on the amount of space, and read/write IO available to the system for the trauma signals while the mathematical model does not, though developing mathematical models for a clinical signal requires medical studies of the signal and all identifiable symptoms in the signal.

Chapter 3

3. System Requirements and Architecture

3.1 System requirements

This section exhibits the system requirements, following by an actor classification,functional requirements , use-case diagrams, non-functional requirements and the result of the first phase of prototype development

3.1.1 Requirements Abstract

The definition of the system requirements can be divided into three main parts. First, evaluating the system's main goals, defining the scope and boundaries. The second part involved researching into other projects and studies with similar aspects. Mostly to discover possible technology answers to specific problems.

Finally, numerous interviews were made with project supervisor to further design the system, how it should work and possible alternatives to problems. An overview of topics appointed by the supervisors::

- A system that return clinical signals
- The system should be able to create a stream of data instead of a wait until completion
- Web Application, to interact with the system
- Save after interacting with the system
- Review saved signals
- Export/Import signals
- Access to the system without using the Web Application
- Mobile App to reenact as a peripheral emitting signals

3.1.2 Context Description

This subsection exhibits a description on how the system is expected to work and be used by different stakeholders.

Starting things off, the system must wait for a request of data, and within the request there should be pre conditions to tell system what type of data it should generate. The system must be able to generate and accept requests at the same time. Data can be generated at a high frequency like 1kHZ and a length upto two hours, thus the system must be able to pause and resume data generation when needed. The system is only interacted through authenticated connections.

On the other end, data analyzer -- someone who develop studies and papers through large amounts of data. The acquisition of signals should be fast and responsive as soon as a request is made through the Web application. The data analyzer should be able to define preconditions and apply modification to the signal at any given time. They should also be able to replay their previous signals and download it into a familiar format.

The programmer -- someone who integrates the system into their own projects. The programmer should have a section where they can view possible mechanism to interact with the system, documentation to each mechanism and an example to get started. All requests made outside of the platform should be authenticated.

3.1.3 Actors

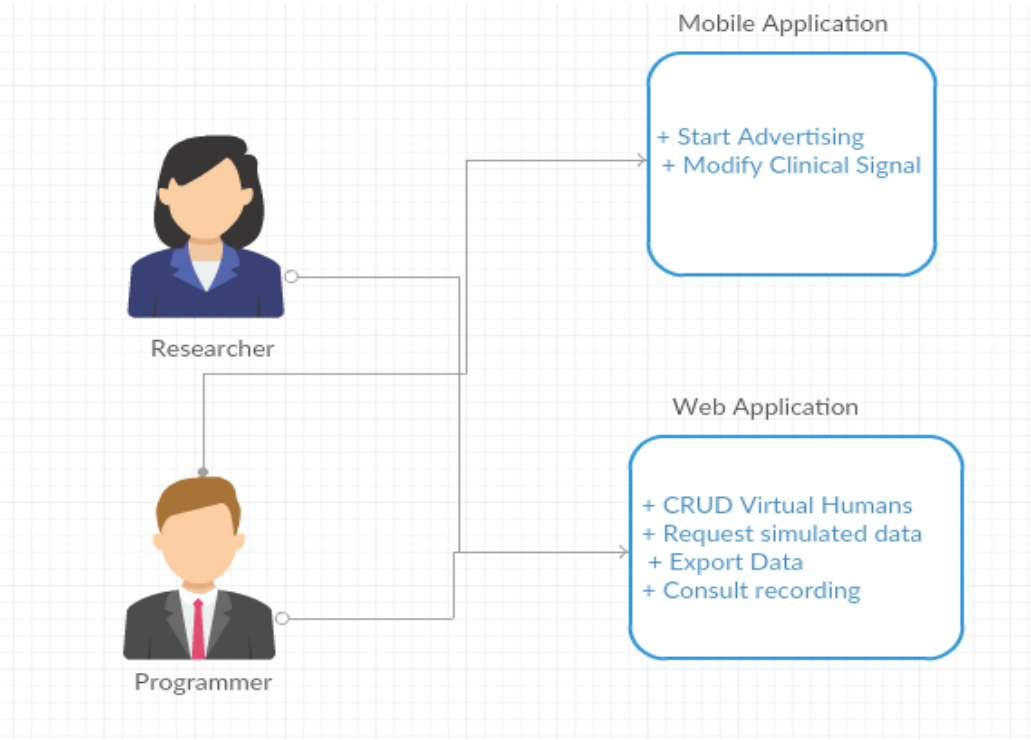
The target users for the system are mainly researchers and developers, whose goals are to further study and implement solutions that need clinical signals. Though the use of the system can vary, tools and interface needs to be consistent and not ambiguous.
The main actors are:

- Researcher: User from a specific field with a primary intent to collect and analyse data. Access to creating/saving/deleting/downloading signals through the web application. Actor should identify the tools available to manipulate signals.

- Programmer: Inherits all the aspects of a researcher. Allowed to access the system outside the web application.

3.1.4 Use Cases

The next figure represents the use case model of the entire system, splitting into two main components: mobile application package and the web application package.



10 Use Case

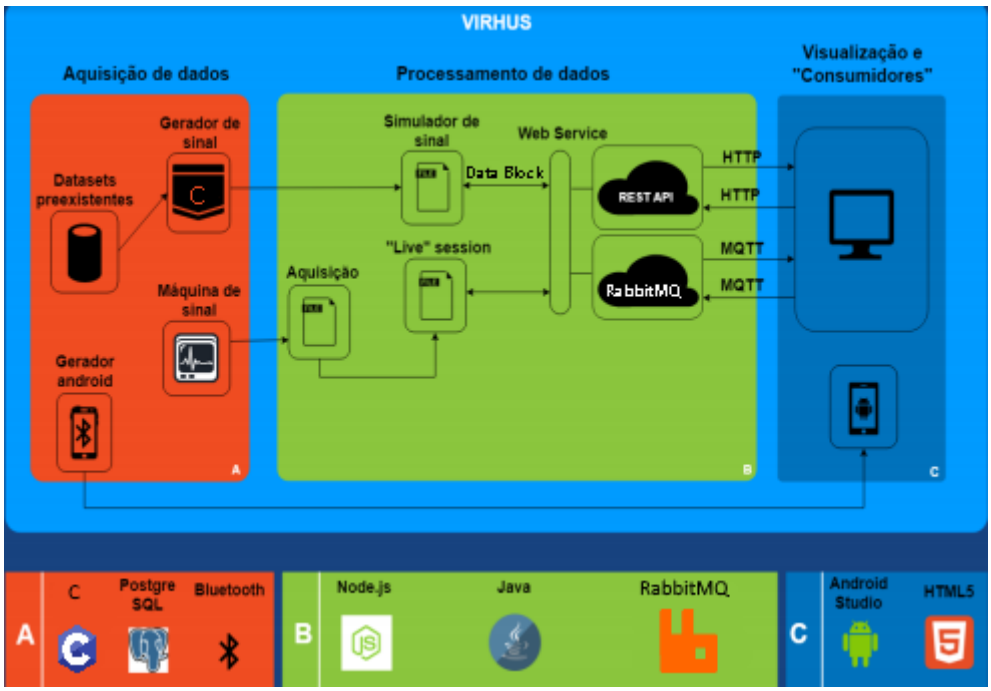
3.1.5 Non-functional Requirements

In this segment presents a list of non-functional requirements and a description of a property or attribute that the system must meet or condition that it must respect.

- Reliability: When a user performs a data request, the system should never drop the connection at any point, error-handling must be treated carefully and in case of system errors, the system should notify the web application to avoid misunderstandings and abandoning of the system.
- Security: Creating an API and allowing programmers to integrate into their project is good but there should be a level of tolerance of the amount of requests made per account.
- Efficiency: Since clinical data can range from seconds to hours and reaching upto 1kHZ, the server's resource should be carefully used. On the client's end, rendering serie graph should be fast and fluid, consuming and overloading the device is not desirable.
- Portability: Users will access the web application through a network connection, regardless of the browser, it should properly work.

3.2 System architecture

This section presents an overview of the system architecture and the technological model.



11 Virhus Diagram

Figure 11 represents an overview of all technologies used within the system.
The red area represents the backend of the system, where the signals are generated. The green area represents the data being processed and how it can be accessed, through the REST API or MQTT. The Blue area is technology of the consumer, what they need to access and interact with the system.

Chapter 4

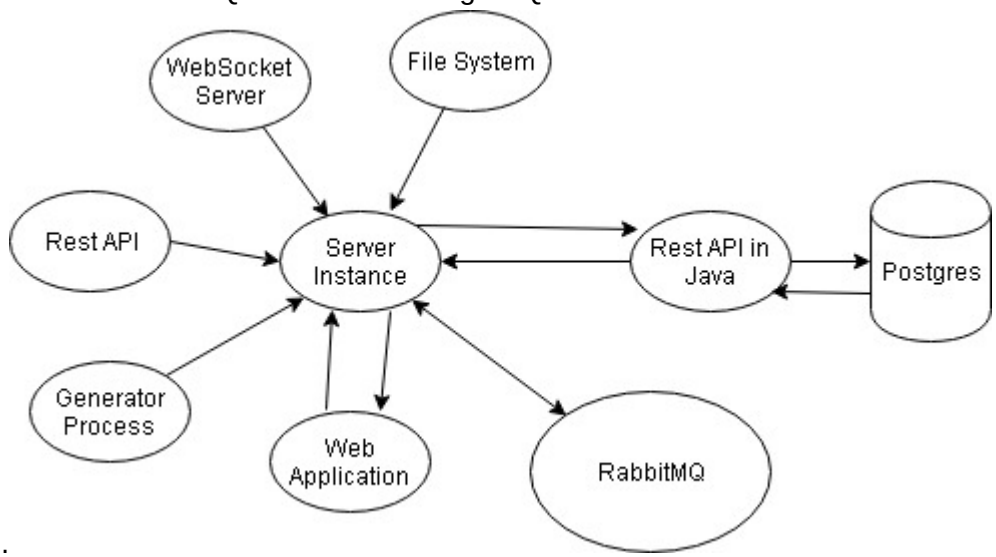
4. Implementation

4.1 VIRHUS Implementation

In this section provides an overview on the implementation of VIRHUS System, related Rest API's and the mobile application. Describing in multiple phases, backend of VIRHUS, Web Application, VIRHUS API, Messaging mechanism and ending with the BLE Peripheral Application.

4.1.1 System

The system is composed of several roles, Rest API,Web Application, WebSocket Server, Data storage/manipulation and generator controller. Also uses client based apis to communicate to RabbitMQ and Rest + PostgreSQL.



.

12 WebSockets Diagram

Creating star-model where everything passes through the server. This methodology enforces all connections and requests to go through the server, protecting any possible attack or breach of data.

As mentioned in section 2.2.1, Node.js is on the rise, and the demands of professionals is significantly high, also considering performance and efficiency that Node.js brings. The decision was made and the server instance implementation would be based on Node.js.

4.1.2 Generator

ECGSYN, in section 2.1.1, showed a valuable way to generate clinical signals from thin air, and also offering a possible method to generate health conditioned signals from it. Since the open source software was freely available under the GNU GPL(General Public License), integrating such work was possible.

ECGSYN was written in C, and the code was very difficult to understand. The code had a misuse of names, low commentary and unused variables. The parameters was quite extensive and repetitive:

```
/* First step is to register the options */

optregister(outfile,CSTRING,'O',"Name of output data file");
optregister(N,INT,'n',"Approximate number of heart beats");
optregister(sfecg,INT,'s',"ECG sampling frequency [Hz]");
optregister(sf,INT,'S',"Internal Sampling frequency [Hz]");
optregister(Anoise,DOUBLE,'a',"Amplitude of additive uniform noise [mV]");
optregister(hrmean,DOUBLE,'h',"Heart rate mean [bpm]");
optregister(hrstd,DOUBLE,'H',"Heart rate standard deviation [bpm]");
optregister(flo,DOUBLE,'f',"Low frequency [Hz]");
optregister(fhi,DOUBLE,'F',"High frequency [Hz]");
optregister(flostd,DOUBLE,'v',"Low frequency standard deviation [Hz]");
optregister(fhistd,DOUBLE,'V',"High frequency standard deviation [Hz]");
optregister(lfhfratio,DOUBLE,'q',"LF/HF ratio");
optregister(seed,INT,'R',"Seed");
opt_title_set("ECGSYN: A program for generating a realistic synthetic ECG\n"
 "Copyright (c) 2003 by Patrick McSharry & Gari Clifford. All rights reserved.\n");
```

13 ECGSYN Parameters

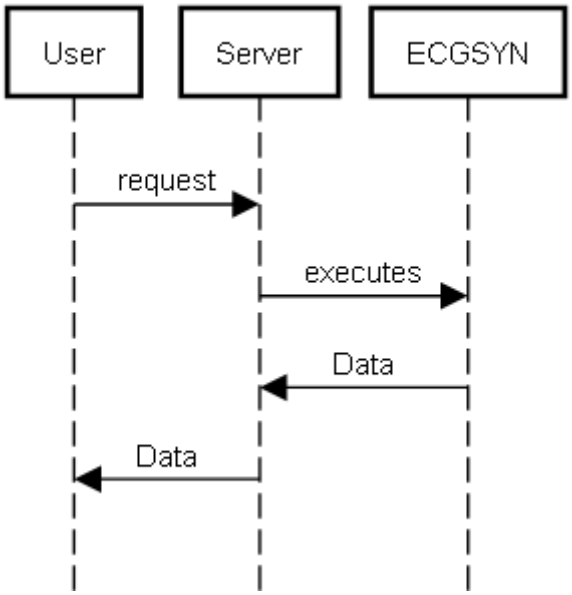
But the overall output is simple and straightforward.

```
0.000000 1.075095 3
0.003906 1.031029 0
0.007812 0.916540 0
0.011719 0.748307 0
0.015625 0.549199 0
0.019531 0.342897 0
0.023438 0.149521 0
0.027344 -0.016643 0
0.031250 -0.147374 0
0.035156 -0.239553 0
0.039062 -0.293963 0
0.042969 -0.314364 4
0.046875 -0.306833 0
0.050781 -0.279048 0
0.054688 -0.239311 0
```

0 (no peak)
1 Top of P-wave
2 Q-point
3 Peak of R-wave
4 S-point
5 Peak of T-wave

14 ECGSYN Output

4.1.2.1 Integration, How the system interacts with it

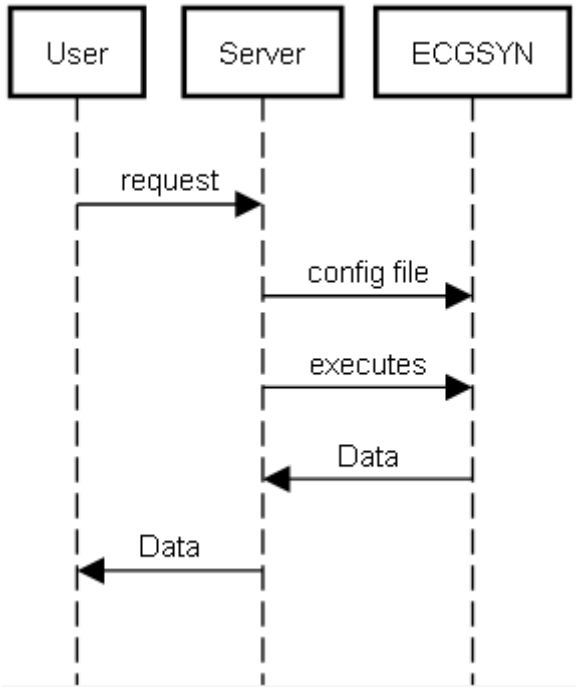


15 System Interaction Sequence

The Figure above displays a linear solution in adding ECGSYN to the system. Decoupling the User from the generator and becoming flexible with the data.

The output on the other hand provided too much information, generic labeling and rr timestamps. The excessive data consumed too much resources from the system and had to be adjusted. Removing both columns made data handling much easier.

Another issue regarding the program was the amount of parameters it had, though inside the code the global variables have default values assigned to them, this style of use made lines of executions extensive and highly dependente.



16 System Interaction Sequence(2)

A different approach was needed, thus developing an important entity to the system, config files. Config files are made up of the various parameters needed to run ECGSYN.

Example of a config file:

```
-60.0 -15.0 0.0 15.0 110.0
1.2 -5.0 30.0 -7.5 0.5
0.15 0.1 0.1 0.1 0.3
50
128
128
0
60
0
0.1
0.25
0.01
0.01
0.5
1
50
100
200
1
ABOVE ORDER!!!
ti PQRST this moves spikes on X axis
ai PQRST this moves spike on Y axis, from positive to negative
bi PQRST this change size of spike on Y axis
aprox number of heartbeats
ecg sample in frequency
internal frequency
Additive noise
heartbeats per minute
heartbeat deviation
low frequecy
high frequecy
low frequency deviation
high frequency deviation
lowfreq/highfreq ratio0
seed, used to add noise every X unit of data
total time, delay of (1000/ecgfreq)*total_time ms
```

17 Config file, represents the Virtual Human in the system

ECGSYN suffered modifications:

- added an option to load a config file
- Addition methods to properly read the file
- Adding a delay option to slow down the flux of data
- Implementing a more realistic noise

4.1.2.2 Other Signals, Import Pre Existing Data

Mentioned in section 1.2, the system supports other clinical signals. Since there were no algorithmic solution, critical decision was made to import clinical signals to project. These clinical signals were supplied by the supervisors of the project. It was database of experiments where patients were undergoing a stimulation test. There were 3 types of tests, Neutral, Happy and Fear. Each experiment had 2 phases, baseline which provided data about the patient and stimulation where patient was presented a video to emulate Neutral, Happy or Fear. The experiments were measuring ECG; EMGM, EMGZ and EDA and the data following the csv format.

Since integrating the ECGSYN was a success, the decision made here was to crop out the ECG column from the data given and replace it with the generated version. Acquiring 3 distinctive stimulations, this became a functional feature, where the user could later adjust the stimulation, creating unique signals. For this to happen, the data needs to be properly label, introducing a fifth column which defines the stimulation of the point of data.

Example:

```
64

1.065276,0.0039673,-0.005188,1.3791,neutral

0.527767,0.02533,-0.03479,1.3791,neutral

-0.205024,-0.0033569,-0.020752,1.3791,neutral

-0.400000,-0.029907,0.031738,1.3794,neutral

-0.251097,-0.012817,-0.014954,1.38,neutral

-0.141936,0.0012207,-0.041504,1.3797,neutral

-0.118427,0.0094604,-0.031738,1.38,neutral
```

18 System output data

4.2 Rest API, within the Server

In this section, an overview of the implementation of the Rest API.

4.2.1 Rest API

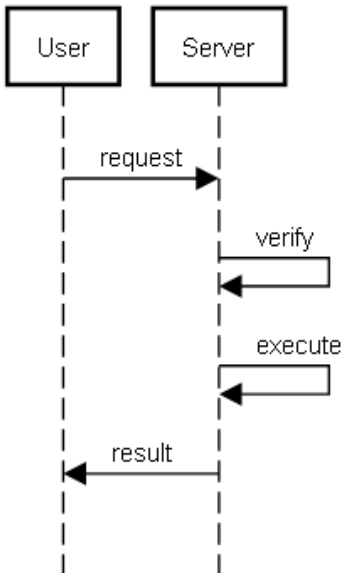
The purpose of this Rest API is to host the Web Application of this project, handle the authentication and creation of Virtual Humans.

Looking back to the use case in section 3.1.4, the image presented functional requirements to the system:

- CRUD Virtual Humans
- Request simulated data
- Export Data
- Consult recording

4.2.1.1 CRUD Virtual Humans

Create, Read, Update, and Delete Virtual Humans, the implementation behind these features we quite straightforward



19 CRUD Sequence

The verify in this context refers to the process of detecting possible system errors and attacks towards the system. Since these requests are coming from an unknown source, the system should always check before execute guaranteeing security and reliability within the system.

4.2.1.2 Request Simulated data

The implementation behind this feature will be handled in section 4.2.3 as the Rest API does not handle any type of request from this feature.

4.2.1.3 Export Data

Very linear sequence where an authenticated user asks the system for an exportation of data, and is immediately delivered. Just like the sequence diagram 19 in section 4.2.2.1, the system first checks if the data exists and belongs to the user to ensure confidentiality and security within the users.

4.2.1.4 Consulting Records

The implementation behind this feature will be handled in section 4.2.3 as the Rest API does not handle any type of request from this feature.

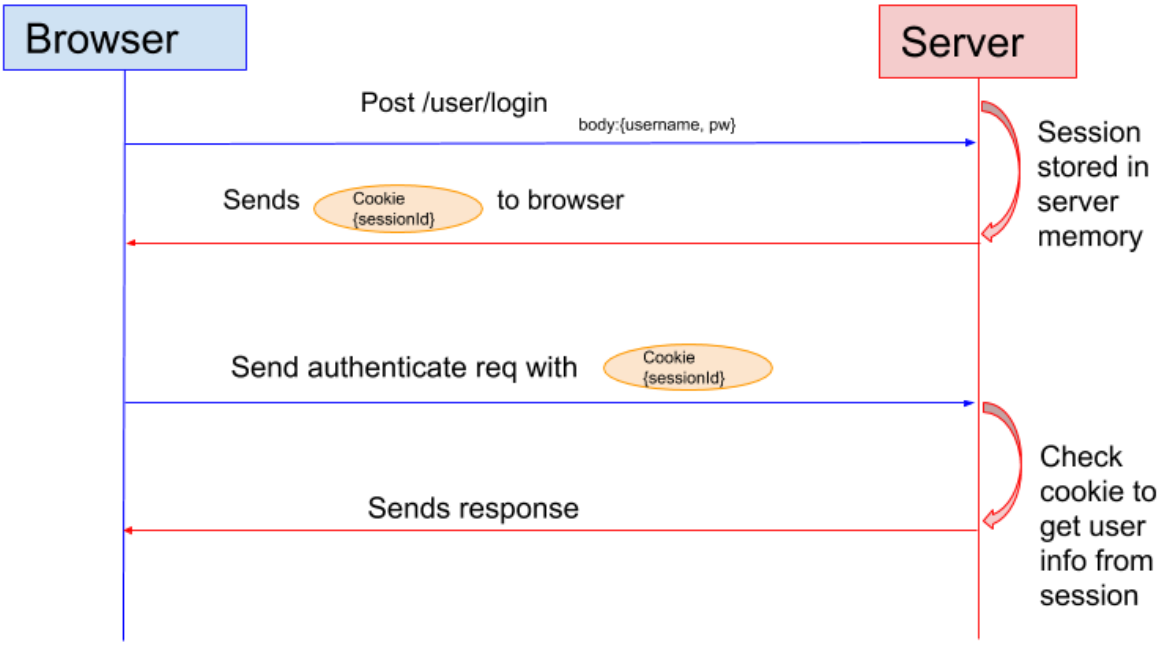
4.2.2 Authentication

Authentication is used for identifying users and provide different access rights and content depending on their id. In most cases the application provides a login form with credentials to verify the user. In the implementation, the tools used were:

- express ()
- express-session (to handle session)
- body-parser (for parsing incoming requests)

4.2.2.1 Session Based Authentication

In the session based authentication, the server creates a session for the user after logging in. The session id is stored on a cookie on tge user's browser While staying logged in, the cookie, subsequent requests are sent along with the cookie. The server then compares the session id against the session information stored in memory to verify the user's identity.



20 Authentication Session Sequence

4.2.3 WebSockets

WebSockets is a computer communications protocol operating at layer 7 in the OSI model and depend on TCP at layer 4. WebsSockets is designed to work over HTTP ports 80 and 443 as well as to support HTTP proxies and intermediaries.

The purpose of WebSockets in this project is to support data requests from the user. Data from the generating specifically. WebSocket enables direct interactions between a web browser and a web server providing real-time data transfer from both ends. Since the communications are done over TCP port 80/443, is a benefit for those environments which non-web internet connections using a firewall.

In Node.js there is a tool called Socket.io which simplifies the complexity in using websockets in the first place. Socket.io is a Javascript library that provides real-time communication with the server, although it can used as a wrapper for WebSockets, it provides useful features like:

- Broadcasting to multiple sockets
- Storing data associated with each client
- Asynchronous I/O

4.2.3.1 Connections

Before jumping into how the data is transferred, requests that are made to the server needs to be authenticated and validated. In the previous section, the authentication session verified

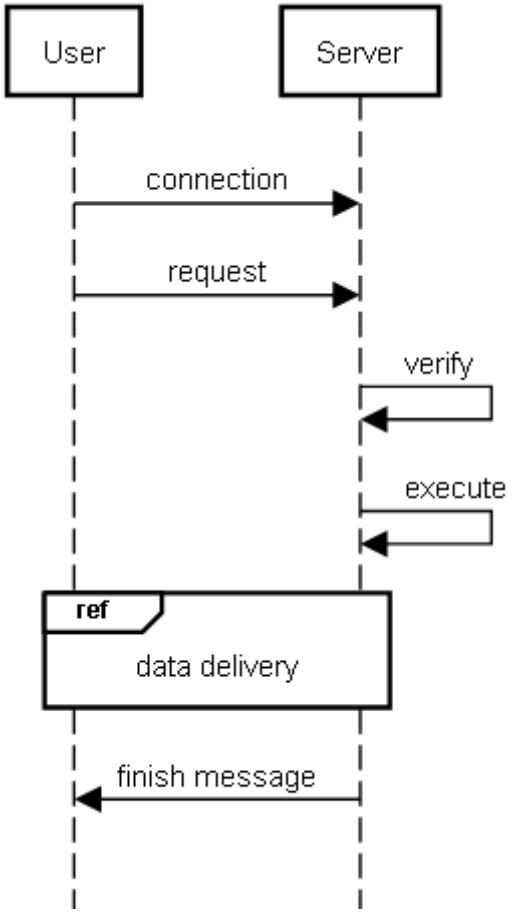
all incoming requests to the api. For WebSocket connections, it is possible to use the express session middleware as a Socket.io middle with a small adapter like the following:

```
io.use(function(socket, next) {
    sessionMiddleware(socket.request, socket.request.res, next);
});
```

From this, it is now possible to extract the same session id found in Rest API appear in the WebSocket connections.

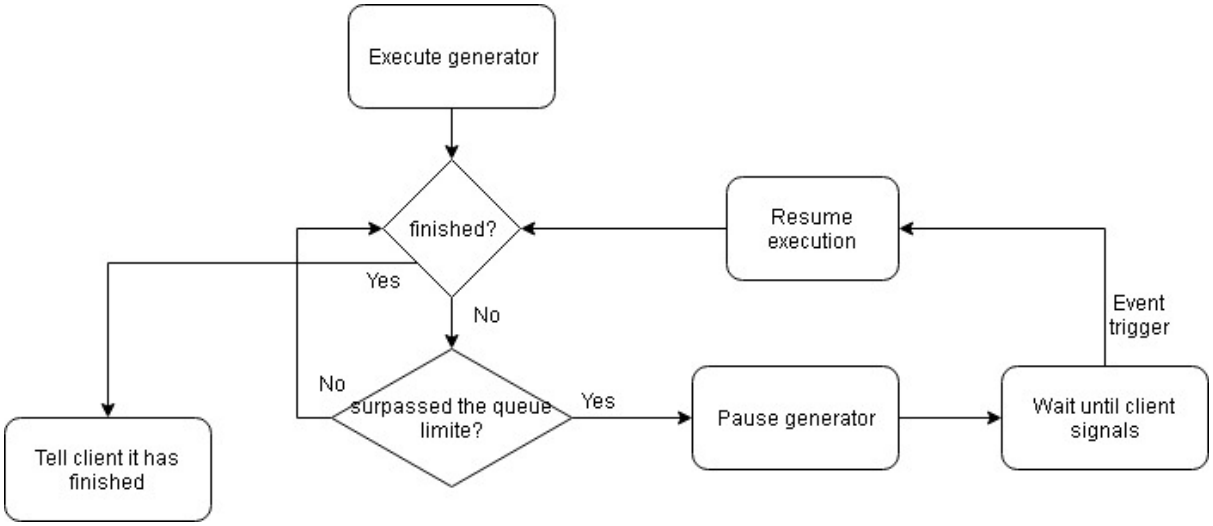
4.2.3.2 Data Transfer

The implementation behind how data is transferred in the system goes by the following:



21 Request Sequence

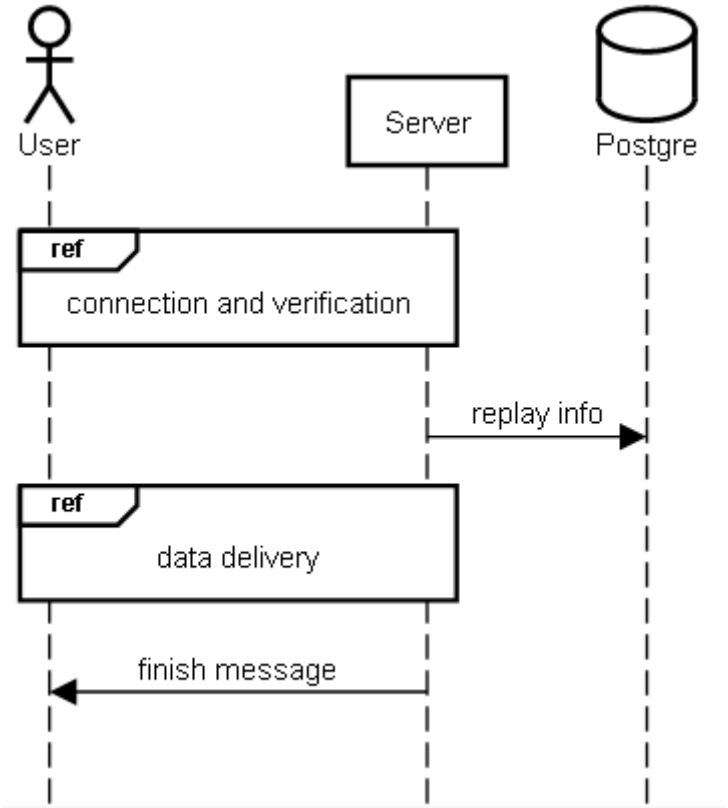
The client first initiates a connection with the server and then follows by a request asking to simulate a Virtual Human or replay an existing Recording. This process can occupy large amounts of resources from the system, and cause reliability issues to the system. So a limited queue had to be implemented causing the process where the generator is running shift to a paused state. Saving resources from the system all together.



22 System workflow

4.2.3.3 Data saved

In the requirement diagram, there is a feature where the user should be able to replay previous signals made through the web application. These replays must exist within the system domain.



23 Data saved Sequence

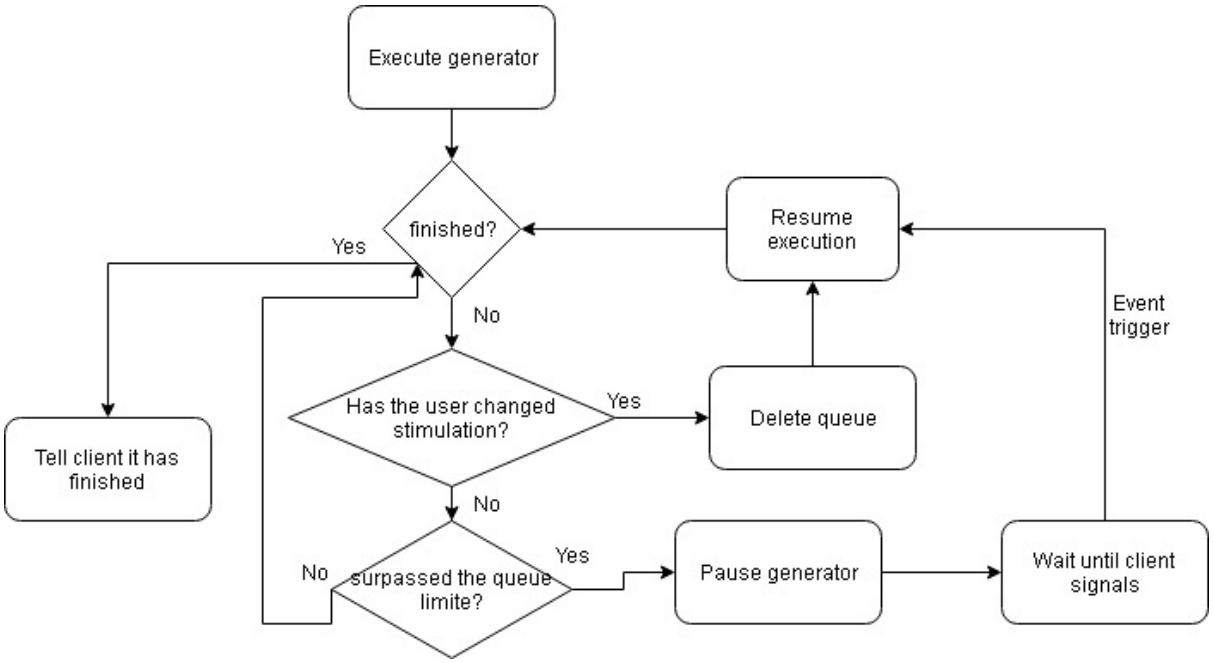
By saving the location and Virtual Human associated to the replay, the system is able to quickly fetch e deploy replays as if the was generating the data.

4.2.3.4 Data Manipulation Mechanism

As previously stated, the clinical data given by the supervisors had three unique stimulation Neutral, Happy and Fear. The challenge here was to develop a mechanism that allows the user to quickly alternate between stimulation through the generating process.

The solution to this problem was quite simple, considering that the only clinical signal to worry about would be the ECG signal as it contain PQRST format.

Jumping back to the generator, after a close inspection in the output and code, the conclusion was in the fact that the generator's first point of data was an R spike and the last point of data was before an R spike meaning that the data was being generated accordingly to the frequency number. So for example, data generated at a frequency of 64, every 64th unit was an R spike.



24 System workflow upon change of emotion

In the following implementation, throughout an on going data generating process, if the user decides to change the stimulation of the Virtual Human the system will dump data after position X which is the predefined frequency and concat data with the new stimulation.
The only downside to this that the user will only notice a change after a couple seconds of data.

4.2.4 Virhus API and VirHus.JS

In this section is an overview of the implementation of the API and Virhus.js

4.2.4.1 Virhus.js

Virhus API is designed to help developers integrate Virhus Solution into their project, offering Virhus.js, javascript class that connects and interacts with the system. The class depends on Jquery.js and Socket.io to properly work.

The entire manual is available on the web application, under documentation for more examples.

```
<script src="/js/jquery.js"></script>
<script src="/socket.io/socket.io.js"></script>
<script src="/js/virhus_api.js"></script>
<script>
  $(document).ready(function(){
      var virhus = new Virhus();
  });
</script>
```

25 Virhus.js example

Virhus.js

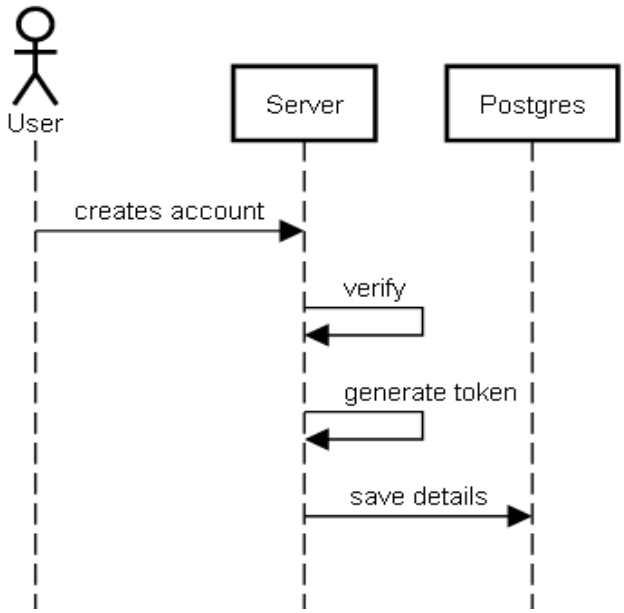
Events	Description
'freq'	The event 'freq' is emitted when the server is publishing the frequency of the following data.
'data'	The event 'freq' is emitted when the server is publishing a data chunk.
'close'	The event 'close' is emitted when the stream and any of its underlying resources have been closed.
Methods	Description
.connect('ip:port','token')	The method .connect() is used to establish a connection to the websocket server. Token is used to authenticate.
.disconnect()	The method .disconnect() is used to disconnect from the WebSocket server.
.downloadCSV('file.dat')	The method .downloadCSV() is used to download a target data file.

.get_config('VirtuHuman',callback)	The method .get_config() is used to get configuration and records of the target Virtual Human.
.isConnected()	The method .isConnected() is used to check if the WebSocket connection is still open.
.requestData('file.dat')	The method requestData() is used to request an existing data from the server
.playRecord('file.dat')	The method.playRecord() is used to start an interactive stream with the server.
.play('virtualHuman')	The method .play() is used to start an interactive stream with the server.
.stop()	The method .stop() is used to stop the current interactive stream with the server.
.whoami()	The method .whoami() is used to request current session details from the server.

4.3.4.2 Token Authentication

Token based authentication works by ensuring that each request made to a server is accompanied by a signed token which the server verifies for authenticity and only then respond to the request.

The system uses a tool called uuid-token-generator that provides a class that generates random tokens with custom sizes and base-encoding using RFC 4122 v4 UUID algorithm. User obtains a token only after creating an account on the web application.



26 Token Authentication Sequence

4.2.5 RabbitMQ

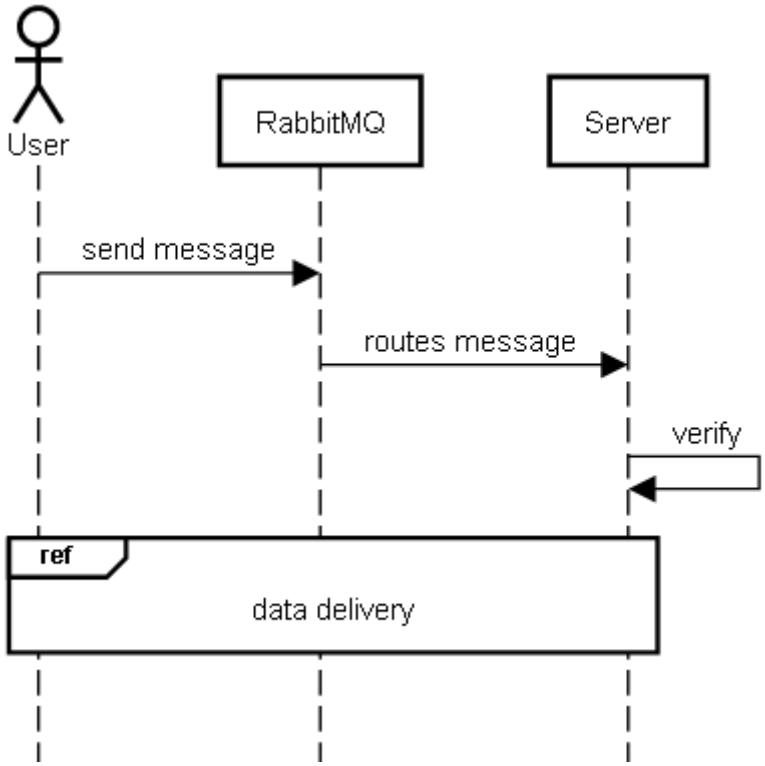
This section gives a brief summary the integration of RabbitMQ in this project, configurations and how it interacts with the system.

4.2.5.1 RabbitMQ

As referred to in section 2.3.5, RabbitMQ is message-broker designed to receive incoming messages from applications and perform any routing/ modification on them. Thus decoupling end-points, which facilitate reuse of intermediary functions.

The purpose of adding RabbitMQ to this project allows developers and researchers the ability to access the system, integrate RabbitMQ client as an alternative to the Virhus.js and to perform data analysis.

To interact with the server, RabbitMQ client must first connect to RabbitMQ broker and send a message in the main queue containing the VirtualHuman and a recording. Once the message is routed to the server, it immediately opens up a queue with VirtualHuman-File.dat, and start sending data at the specified frequency speed.



27 RabbitMQ client interaction Sequence

4.3.5.2 Configuration

The default configurations on RabbitMQ gave existing queues the ability hold infinite amount of repetitive data, this problem was solved easily by creating a policy where all queues had a max-length of zero.

28 RabbitMQ Policy

4.3 Continuous Integration & Continuous Deployment

Continuous Integration is the practice of merging all developers work into a shared repository. When developing a feature or change, the developer takes a copy of the current code base to work with. These changes are locally tested and submitted to the repository.

Continuous Deployment is an approach in which software functionalities are delivered frequently through automatic deployments. Usually involves tools that automatically builds the project, perform unit tests and push the code to the hosting machine.

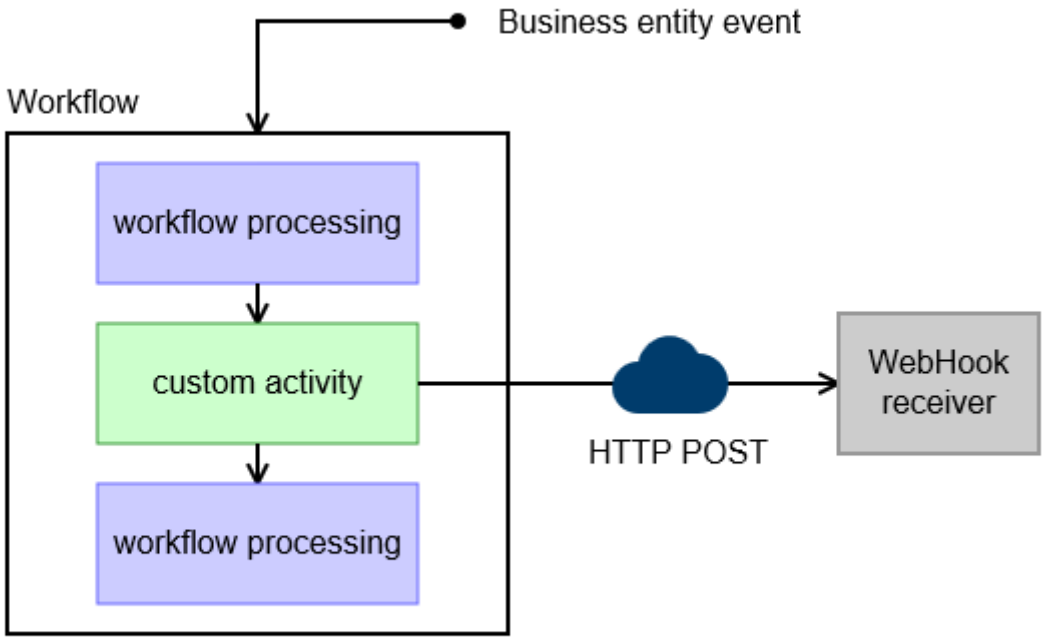
The tools used here were:

- Jenkins; automation server that contain numerous amount of useful plugins to help automate the software development process
- Gitlabs;web based DevOps lifecycle tool that provides a Git-Repository
  In this project,

4.3.1 Git Webhooks

Webhooks allows to build and set up integrations which subscribe to certain events to the Git repository. When one of those events is triggered, Git sends an HTTP POST payload to the webhook's configured URL. It was impossible to use webhooks in this project due to the location of the virtual machine. Hosting machines within the University can only be accessed

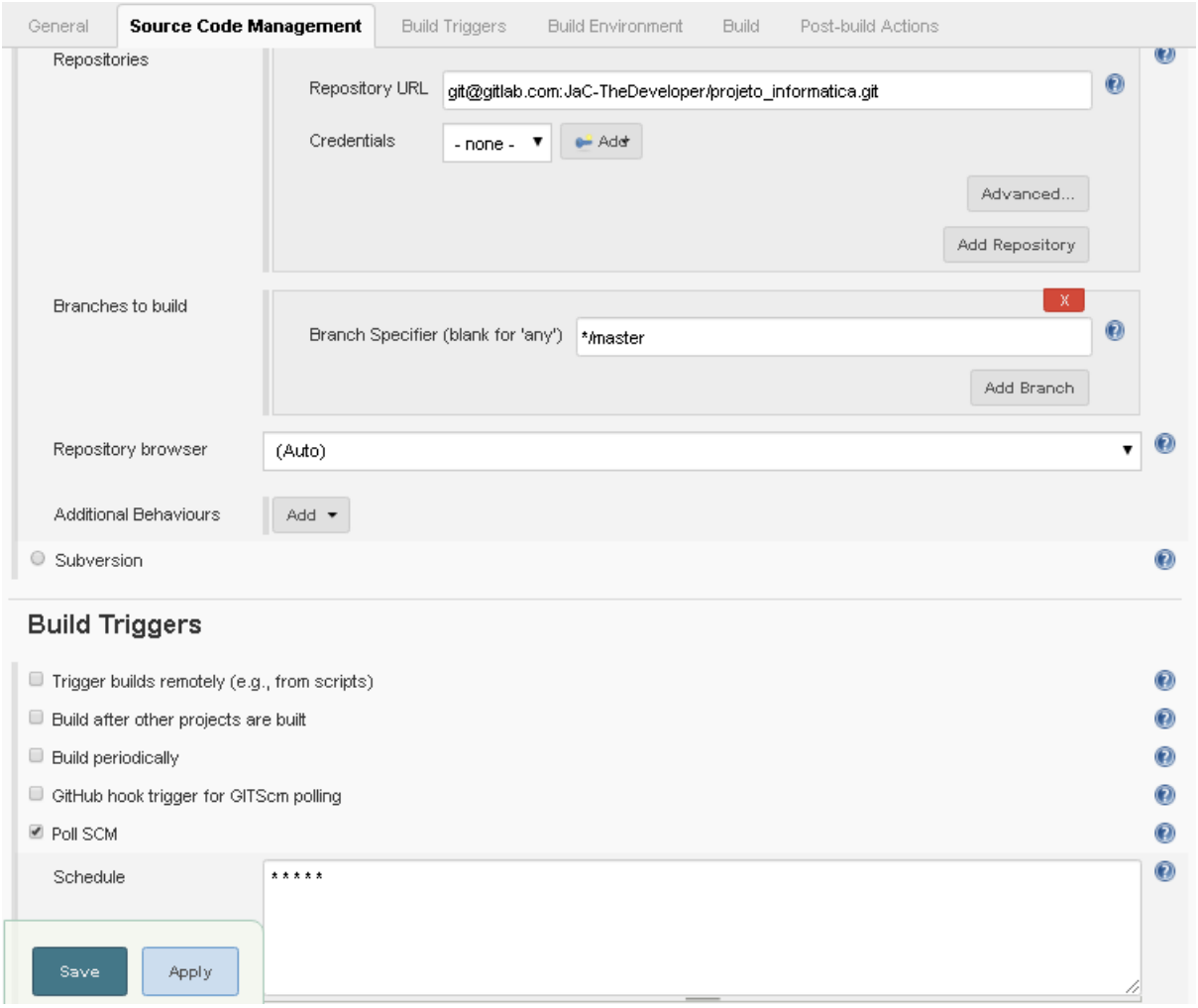
within the network or through the official VPN of the University. Despite of this, webhooks is still a recommended mechanism for building and integrations



29 Git Webhooks Diagram

4.3.2 Jenkins setup

As mentioned in section 4.3, Jenkins contains a variety of plugins and tools that helps build and set up integrations. Inside Jenkins there is a plugin called Git Plugin which allows the use of Git as a build SCM.
Here is configuration used



30 Node Config

4.3.3 Poll SCM

Since Webhooks does not work inside the University network, Poll SCM an alternative that's constantly checks the repository for any updates. “* * * * *” tells Jenkins to poll every minute.

4.3.4 Mocha Unit tests

Mocha is a Javascript test framework running on Node.js, allowing for flexible accurate reporting, while mapping uncaught exceptions to the correct test cases.

Since the core of the project is in Node.js, Mocha was excellent addition for unit testing and automatic builds.

A simple example of testing the API:

```
var request = require('supertest');
var app = require('../index.js');

describe('GET /', function(){
        it('respond with test', function(done){

                request(app).get('/test').expect('test', done);
        })
});
```

31 Mocha unit test

4.3.5 Build and Deployment

As mentioned previously, Jenkins can build and deploy projects through nodes. A node represents a project or feature that perform various tests, from unit to UI testing.
In this project there are 2 scripts:

- Test; contains Mocha and the test folder
- Deploy; containing bash commands to properly publish the successful code to the virtual machine.

4.4 Android App

This section gives a brief information about the Android App developed for this project. We started to create a connection between 2 Android phones with 2 different Apps via Bluetooth, one worked as the client and the other as the server.

The first app (client app), is the one where the user can see every data recorded from the second app (server app), he decides the device he wants to connect via Bluetooth from the list of previous paired devices. Then he can start the simulation, where he will se 5 different fragments containing different data, each one exclusive from all the signals, with one extra signal different from the main project, Heart Beat.

The second app (server app), only has one Activity that can accept the connection with the other app. It has the data from all the signals and it sends via Bluetooth to the client app.

5. Conclusion

5.1 Project Summary

The project started with the idea of replacing physical,expensive and limited signal generators which made signal acquisition extremely hard for research developments. Although some of the projects presented in chapter 2 tackled this problem, none of them flexible, scalable or available online. This made Virhus a promising system, ready to dominate the market interest in data acquisitions for development.

As mentioned a lot in this report, entities in research department would heavily benefit from Virhus, due to having access to an unlimited amount of data. But there are also entities in the medical department, like ESSUA, which would highly benefit this system and the system from them. ESSUA is a branch of the University of Aveiro, where all the medical courses are lectured. Throughout the classes, with what Virhus brings, teachers can opt to a virtual environment to supply objective clinical data to their students. The medical equipment in this institution could sometimes draw misleading information, the detection can be done by applying Virhus's data to assess and maintain.

The project had a rough beginning, the first interviews with the supervisor made the overall comprehension poor and almost impossible to figure out functional features, a system architecture. On top of that, the knowledge required to understand clinical signals was critical since the entire system operated on these signals. It took a couple of months of research, creating the required knowledge overall. Due to the time used, implementation, testing etc had to be done outside of the curriculum schedule.

5.2 Conclusions

Virhus had its ups and downs throughout the implementation, one its downs was with the inclusion demand of other signals which then further spread into stimulations. Causing alot of confusion and rollbacks in the code. When the project started, it was not quite clear how and what the data should be presented, many mistakes were made and further fixed after various iterations with the supervisors.

One of the positive upsides was around the optimization of data being transferred. In the early stages of the project there were a lot of disappointing results in rendering graph series on the web application. This was due to the velocity of generator, producing large amounts and overflowing the memory limit of the users computer. Not only the generator was faster the web

application, the data that became unviewable in the web application still existed provoking reliability and feedback issues in presenting the data. User at a certain point thought they lost connection to the server.

5.3 Future work

Future work on this project should start by adding more clinical signals of other stimulations, increasing the creativity. Create a soft transition when changing between stimulations and implement a signal analyser on the UI where the user can see certain timestamps of the signal. Since the world of technology is constantly evolving, the API class and the messaging client will also need maintenance from time to time, guaranteeing functional integration with other projects.

Appendix A

1. Command to run the server

```
node index.js < /dev/null > /dev/null 2>&1 &
```

2. Perform unit tests

```
./node_modules/.bin/mocha ./tests/tests.js
```

References

Clinical Simulations, Ambulance in a box
https://books.google.pt/books?id=gonEANOwR-gC&pg=PA439&lpg=PA439#v=onepage&q&f=false

Physionet A Realistic ECG Waveform Generator ECGSYN
https://physionet.org/physiotools/ecgsyn/

Biomedical Engineering Systems and Technology by Mauricio Tavares
https://link.springer.com/book/10.1007/978-3-540-92219-3