

**Faculdade de Engenharia da Universidade do Porto**



# **Neuroscientist-Friendly Seizure Analyzer Application for Epilepsy Monitoring**

**Beatriz Neves Garrido**

DISSERTATION

Mestrado Integrado em Engenharia Eletrotécnica e de Computadores

Supervisor: João Paulo Cunha, PhD

July 28, 2020



# Resumo

Pacientes que sofrem de doenças neurológicas, como epilepsia, podem ser diagnosticados e tratados usando um exame de eletroencefalograma, no entanto este teste é avaliado através de uma observação visual e, por isso, é feita uma análise subjetiva. Portanto, a quantificação de movimento pode ser considerada um aliado útil na avaliação destes indivíduos. Como resultado, o sistema NeuroKinect foi previamente desenvolvido pelo laboratório hospedeiro, para ajudar a medir o movimento humano e favorecer o trabalho de especialistas em neurologia no tratamento de seus pacientes.

Este sistema era relativamente recente e apresentava alguns problemas de desempenho e usabilidade. Portanto, houve uma necessidade urgente de melhorar o software. Dito isto, este estudo focou-se na criação de uma interface para corrigir aquela anteriormente desenvolvida. O produto resultante precisava de ter um tempo de resposta rápido, e ser atraente e claro o suficiente para ser utilizado por qualquer profissional médico.

Foram pesquisadas um conjunto de ferramentas com o objetivo de concluir qual a melhor solução para lidar com o problema dado. Isto inclui não apenas ferramentas de desenvolvimento de interface, mas também criação de comunicação de software, com o objetivo de armazenar as informações obtidas de futuras interações com o utilizador.

Pode-se dizer que os resultados alcançados neste estudo foram satisfatórios, devido ao rápido comportamento e tempo de resposta da interface e da API desenvolvida. Mais importante ainda, o objetivo de facilidade de uso foi alcançada, tendo em consideração o feedback positivo que os utilizadores deram ao interagir com o novo aplicativo, concentrando-nos principalmente nos comentários que os médicos fizeram sobre a interface uma vez que são os clientes mais importantes a agradar, considerando que são o público-alvo deste sistema. Além disso, os testes funcionais realizados mostram que o software melhorou em velocidade e eficiência.

Embora o objetivo principal de desenvolver uma melhor interface tenha sido alcançado, algumas características ainda precisam de ser implementadas, principalmente a ligação entre o front-end e o algoritmo de rastreamento, a fim de calcular métricas reais e examinar o movimento humano dos indivíduos participantes. No entanto, este estudo deu uma contribuição positiva em termos de interface do utilizador e experiência do utilizador para o software inicial, aproximando-se, assim, de alcançar a melhor solução possível para a avaliação da epilepsia.





# Abstract

Patients suffering from neurological diseases, such as epilepsy, can be diagnosed and evaluated by using an electroencephalogram exam, but this test relies on visual observation and subjective analysis. Therefore, movement quantification can be considered a useful ally when assessing these individuals. As a result, the NeuroKinect system was previously developed by the host lab, in order to help measure human motion and favor the work of neurology specialists when treating their patients.

This system was fairly recent and had some issues with performance and usability. Therefore, there was an urgent need to improve the software. That being said, this study focused on creating an interface to correct the one previously developed. The resulting product needed to have a fast response time, and be attractive and clear enough for any medical professional to use.

A set of tools was researched in the interest of concluding which was the best solution to resolve the handed problem. This includes not only interface developing tools but also software communication builds, for the purpose of storing the information acquired from future user interactions.

One could say that the results achieved in this study were satisfying, on account of the quick behavior and response time of the interface and the API developed. Most importantly, the user-friendliness goal was achieved taking into account the positive feedback users gave from interacting with the new application, especially focusing on the comments that the medical users gave regarding the interface since they are the more important clients to please, considering that they will be the target audience for this system. Moreover, the functional tests performed show that the software improved in speed and efficiency.

Even though the main goal of developing a better interface was achieved, certain features still need to be implemented, mainly the connection between the front-end and the tracking algorithm, in order to compute real metrics and examine the human motion of the participating subjects. Nevertheless, this study gave a positive contribution in terms of user interface and user experience to the initial software, therefore becoming closer to achieving the best solution possible for epilepsy assessment.



# Acknowledgments

First of all, I would like to thank Dr. Ricardo Rego, for the availability and his contagious good mood.

To all the BRAIN members for the help and solidarity, especially to Tamás Karácsony, for all the patience and support, and Vítor Minhoto for being such a good friend and assisting me every step of the way.

To my boyfriend, for always having my best interests at heart.

Bia



*"The only place success comes before work is in the dictionary."*

- Vince Lombardi



# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Context . . . . .	1
1.2	Problem Definition . . . . .	1
1.3	Motivation & Objectives . . . . .	2
1.4	Document Outline . . . . .	2
<b>2</b>	<b>NeuroKinect Description and Flaws Overview</b>	<b>3</b>
2.1	Epilepsy Assessment . . . . .	3
2.2	NeuroKinect . . . . .	4
2.3	KiSA . . . . .	5
2.4	Problem Description . . . . .	5
2.5	Requirements Outline . . . . .	6
<b>3</b>	<b>State of the Art</b>	<b>9</b>
3.1	Concept Review . . . . .	9
3.1.1	Framework Definition . . . . .	9
3.1.2	HTTP Methods . . . . .	9
3.1.3	API . . . . .	10
3.2	Front-end Development . . . . .	11
3.2.1	Native GUI Toolkits . . . . .	11
3.2.2	Web Libraries . . . . .	14
3.2.3	Conclusion . . . . .	17
3.3	Web API Development . . . . .	18
3.3.1	Web Frameworks . . . . .	18
3.3.2	Data Formats . . . . .	21
3.4	Desktop Applications using Web Technologies . . . . .	23
3.4.1	Electron . . . . .	23
3.4.2	Proton Native . . . . .	23
3.4.3	Conclusion . . . . .	24
3.5	Related Projects . . . . .	24
3.5.1	Neuro Event Labs . . . . .	25
3.5.2	Epihunter . . . . .	26
<b>4</b>	<b>KiSA v3.0 Design and Implementation</b>	<b>27</b>
4.1	Methodology . . . . .	27
4.1.1	Use Cases . . . . .	27
4.2	System Development . . . . .	33
4.2.1	Code Architecture . . . . .	33
4.3	Results . . . . .	35

4.3.1	Main Menu Screen . . . . .	35
4.3.2	Tracking and Labeling Screen . . . . .	36
4.3.3	Analysis Screen . . . . .	43
4.4	Usability Testing . . . . .	44
4.4.1	Conclusions . . . . .	46
<b>5</b>	<b>API Development and Testing</b>	<b>47</b>
5.1	Implementation . . . . .	47
5.1.1	Data Sent . . . . .	48
5.1.2	Documentation . . . . .	48
5.2	Results . . . . .	49
5.3	Functional Testing . . . . .	52
<b>6</b>	<b>Conclusions and Future Work</b>	<b>55</b>
6.1	Overview . . . . .	55
6.2	Future Work . . . . .	56
<b>A</b>	<b>Interface Results</b>	<b>59</b>
A.1	Main Menu . . . . .	60
A.2	Tracking and Labeling . . . . .	61
A.3	Analysis . . . . .	64
<b>B</b>	<b>API Code</b>	<b>65</b>
	<b>References</b>	<b>67</b>



# List of Figures

2.1	Architecture of the NeuroKinect multi-bed system deployed at the University of Munich’s EMU. . . . .	4
2.2	MOI parameters with highest $\rho$ -score from patient’s 42 seizures MOI. . . . .	5
2.3	KiSA’s graphics user interface. . . . .	6
3.1	API architecture with HTTP requests . . . . .	10
3.2	Python GUI Programming (Tkinter) . . . . .	12
3.3	a GUI with Python and Qt, using “Fusion” style. . . . .	12
3.4	Qt Designer. . . . .	13
3.5	CubeColourDialog, wxGTK Control Appearance. . . . .	13
3.6	Graphical interface of “TriFusion”. An application made using Kivy . . . . .	14
3.7	GitHub Stats: React vs. Vue vs. AngularJS. . . . .	16
3.8	(Possible) Learning Curve for AngularJS, React and Vue. . . . .	16
3.9	API Frameworks GitHub Stats: Django vs. Pyramid. . . . .	19
3.10	API micro-frameworks GitHub Stats: Bottle vs. Flask. . . . .	19
3.11	Stack Overflow Trends: JSON vs. XML vs. CSV . . . . .	22
3.12	GitHub Stats: Electron vs. Proton Native. . . . .	24
3.13	Nelli Initial Dashboard Design. . . . .	25
3.14	Epihunter Companion app . . . . .	26
4.1	Use Case 2 exemplification . . . . .	28
4.2	Use Case 3 exemplification . . . . .	29
4.3	Use Case 4 exemplification . . . . .	29
4.4	Use Case 5 exemplification . . . . .	30
4.5	Use Case 6 exemplification . . . . .	30
4.6	Use Case 7 exemplification . . . . .	31
4.7	Use Case 8 exemplification . . . . .	31
4.8	Use Case 9 exemplification . . . . .	32
4.9	Use Case 10 exemplification . . . . .	32
4.10	Use Case 11 exemplification . . . . .	33
4.11	Code organization . . . . .	34
4.12	Activity diagram for the menu screen . . . . .	35
4.13	Load new seizure . . . . .	35
4.14	Patient data pop-up . . . . .	36
4.15	Activity diagram for the tracking & labeling screen . . . . .	36
4.16	Top bar . . . . .	37
4.17	Change screen options . . . . .	37
4.18	Patient data display . . . . .	37

4.19	Select edit patient data . . . . .	37
4.20	Patient data filled . . . . .	38
4.21	Tracking process . . . . .	38
4.22	Tracking load bar . . . . .	39
4.23	Right-Leg tracked video . . . . .	39
4.24	Left-Hand tracked video . . . . .	40
4.25	Right-Hand not tracked video . . . . .	40
4.26	Sub-type label for different Types . . . . .	41
4.27	Time highlight when MOI selected . . . . .	41
4.28	ROI with no tracking . . . . .	42
4.29	ROI with tracking . . . . .	42
4.30	Analysis with velocity metric . . . . .	43
4.31	Analysis with jerk metric . . . . .	44
4.32	Survey results from regular users . . . . .	45
4.33	Survey results from medical users . . . . .	46
5.1	API placement in the KiSA architecture . . . . .	47
5.2	POST request sent to API . . . . .	50
5.3	GET request sent to API . . . . .	51
5.4	PUT request sent to API . . . . .	51
5.5	API test results . . . . .	53
A.1	Main Menu Screen . . . . .	60
A.2	Tracking & Labeling Screen - Nothing Selected . . . . .	61
A.3	Tracking & Labeling Screen - Add Selected . . . . .	62
A.4	Tracking & Labeling Screen - Edit Selected . . . . .	63
A.5	Analysis Screen . . . . .	64

# List of Tables

4.1 System Usability Scale Template . . . . .	45
5.1 API documentation . . . . .	49
5.2 Response time for each HTTP method developed . . . . .	54



# Abreviaturas e Símbolos

API	application programming interface
BRAIN	Biomedical Research And Innovation
CSV	Comma-Separated Values
EEG	Electroencephalography
GUI	Graphical User Interface
HSJ	Hospital São João
HTTP	HyperText Transfer Protocol
INESC-TEC	Instituto de Engenharia de Sistemas e Computadores, Tecnologia e Ciência
JSON	JavaScript Object Notation
<i>KiMA</i>	<i>Kinect Motion Analyser</i>
<i>KiSA</i>	<i>Kinect Seizure Analyser</i>
<i>KiT</i>	<i>Kinect Tracker</i>
MOI	Movement Of Interest
MVC	Model-View-Controller
REST	Representational State Transfer
ROI	Region Of Interest
SUS	System Usability Scale
UI	User Interface
UX	User Experience
VENV	Virtual Environment
XML	Extensible Markup Language



# Chapter 1

## Introduction

### 1.1 Context

Epilepsy is one of the most common neurological disorders, affecting from 0.5 to 1% of the world population [1].

Currently, the most common way to evaluate such disorder is by performing and examine an Electroencephalogram (EEG), a non-invasive test used to evaluate the electrical activity in the brain. The detection of patterns typical in epilepsy - changes in the normal activity of brain waves - is the way to diagnose this condition and guide its treatment. But because an epileptic episode is of short duration [2], the best way to evaluate these patients is to observe the moment of the occurrence, therefore, a hospital stay is required to oversee the patient and perform an EEG exam.

Even though this exam is very accurate, it is a time-consuming, cost burden solution that requires an in-hospital setting. Furthermore, EEG analysis is still a subjective task due to relying on using visual observation and personal interpretation. As such, quantification of human motion that occurs during the recorded event is considered a useful tool for assessing patients with epilepsy [3].

Hence, the Biomedical Research And Innovation (BRAIN) team at Instituto de Engenharia de Sistemas e Computadores, Tecnologia e Ciência (INESC-TEC), in partnership with doctors from Munich, has developed an application, Kinect Seizure Analyser (KiSA), that can aid the physician's evaluation of neurological diseases, using computer vision and pattern recognition techniques for movement quantification and seizure detection.

### 1.2 Problem Definition

Even though the application mentioned above has shown potential for clinical decision support, using a non-intrusive and low-cost solution, it is a very recent project [4]. Therefore, it can be still improved in regards to software efficiency, fluidity and portability.

Alongside with the algorithm's lack of performance, it is also problematic the way that the system presents itself to the user. The interface is not very intuitive and may have extensive load times, which can be frustrating to handle.

Because of these issues, a new version of the software is currently being developed, to improve the previous system. Alongside with performance boost, it also needs to be taken into consideration the user-friendliness of the final interface.

### 1.3 Motivation & Objectives

To collaborate with the development of the new software, this study's main objective is to develop a new interface that is intuitive, perceptive and, most importantly, user-friendly. The KiSA algorithm will not yet be incorporated with this thesis, due to the fact that it is being improved parallel to the development of this study, therefore it would not be viable to combine two software that are constantly suffering changes.

After this, it is also required to store the information resulting from the user interaction. Therefore, it is necessary to develop an application programming interface (API) to extract the user's activity and save all the important data in a specific server.

Bearing this in mind, the tools used for this goal will need to take into consideration response time, the flexibility of adding new features, user-friendliness and overall usability. Additionally, because it is desired to place this application on a web page, another focus is effective communication between back-end and front-end, as such, the study of different types of frameworks to deal with a web's server-client messaging is also a concern.

### 1.4 Document Outline

After this introductory chapter, five more chapters are presented.

Chapter 2 gives an overview of the previous system and how it can be improved.

Next, chapter 3 presents different tools and frameworks that could be used to handle the problem at hand, as well as other applications in the market that are used for the same purpose as the system mentioned previously.

Then, in Chapter 4 the new interface developed is described, with insights of how its integration works and analysis of the obtained results.

Finally, chapter 5 includes the API developed, which is the end of the data workflow, to save the data acquired previously. Again, with the analysis of the obtained results.

Finally, in Chapter 6 conclusions are drawn and some suggestions for future work are made.



## Chapter 2

# NeuroKinect Description and Flaws Overview

This chapter reviews important background information, essential to understand the problem at hand. It also overviews the flaws of the existing system and does the requirements gathering for this study.

### 2.1 Epilepsy Assessment

The software that will be analyzed in this chapter has as a main functionality Epilepsy diagnosis and assessment. Therefore, it is important to first understand this complicated neurological disorder.

Epilepsy is a neurological disorder that is characterized by recurrent, short duration seizures, that may vary from nearly undetectable to powerful shaking. The best way to evaluate the patient is to observe the moment of the occurrence, designated acute phase, however, a doctor can not be present at every moment of the patient's day and wait to see how and when a seizure will occur. Furthermore, the report of the vigorous given by the patient might not be as accurate as desirable.

As such, a person with this disorder needs to be hospitalized in order for a specialist to evaluate the acute phase and best diagnose and treat the disease. The way to monitor these patients is by performing an EEG test while in the hospital stay and look for the abnormal activity of the brain.

Nevertheless, the EEG approach is not flawless. The exam requires visual observation and the doctor's subjectivity might lead to a small unwanted error in the patient's assessment. Hence visual inspection of the patient's movements can be a useful ally to EEG analysis and epilepsy treatment.

But again, it is not viable to have a specialist full time observing the patient in order to witness an episode and collect information about it. Consequently, the BRAIN team has developed an application that can record the patient's stay and store the information corresponding to a seizure, so that a doctor can, in the future, analyze just the desirable moments.

Nevertheless, this solution is still recent and can improve its performance and effectiveness. Therefore, as the goal for this thesis is to improve the existing system,

it is important to understand the way it operates and its main features. A study of the said project is shown in this chapter.

## 2.2 NeuroKinect

As mentioned before, the BRAIN laboratory has developed an application denominated NeuroKinect, described in Figure 2.1. It is an inexpensive, portable and non-intrusive solution that assesses human motion and helps to identify and monitor patients suffering from neurological diseases [5, 6, 4].

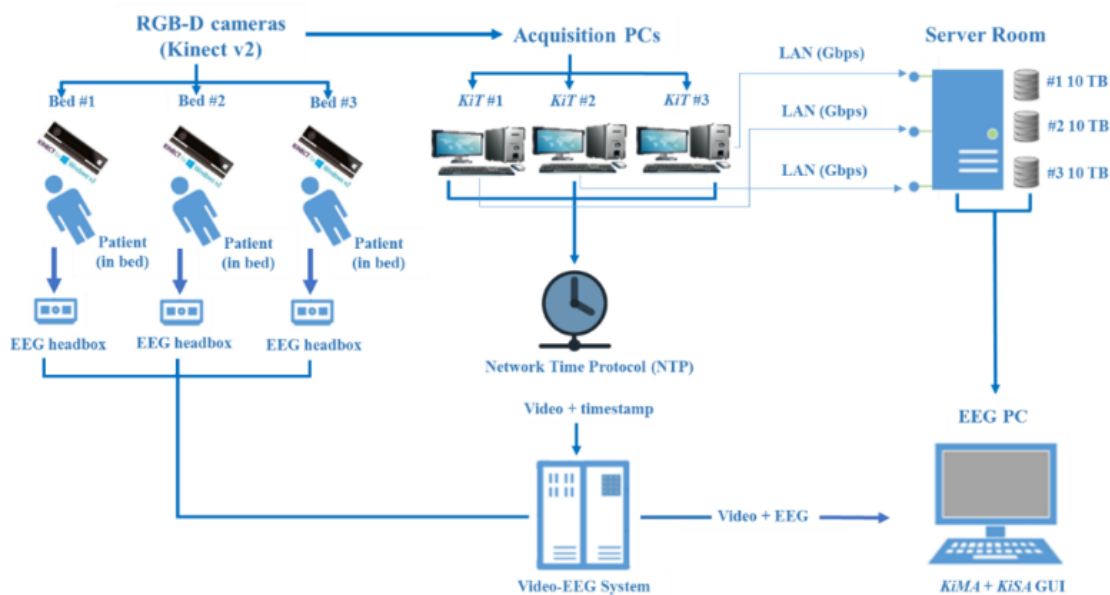


Figure 2.1: Architecture of the NeuroKinect multi-bed system deployed at the University of Munich’s EMU. Adapted from [6]

The workflow of this application starts with the Kinect cameras<sup>1</sup> collecting synchronized video-EEG from the patients. Each camera is then connected to a PC running KinectTracker (KiT), a software that handles all the information acquired.

Later, a supervisor - that can be a clinician or a nurse - manages KiT and defines the initial and final moments of acquisition, *i.e.*, the motion information corresponding to a seizure. The data from these PCs are automatically transferred to the server and is later sent to a database, to be analyzed by the succeeding application: Kinect Motion Analyzer (KiMA) and/or KiSA [5, 6, 4].

The KiMA software is mainly used for gait analysis, it marks and labels specific events and allows for edition and deletion of those labels. The selected information can then be exported for further study.

The KiSA software is more suitable for seizure analysis rather than gait, which is what this study focuses on. Like KiMA, it also provides the ability for labeling the movements of interest (MOI), as well as editing and deleting them. Furthermore, it is

<sup>1</sup>movement sensors developed by Xbox 360 and Xbox One

equipped with a semi-automatic tracking algorithm, used for quantifying the MOI. A more thorough description of this software is described below.

## 2.3 KiSA

This desktop application makes the neurologists' work faster and more efficient, by using a 3D tracking algorithm that quantifies the human motions, to later help in the assessment of patients with epilepsy [7]. The software, developed in Matlab, identifies the parts of the body where relevant movements are present, *i.e.*, Regions of Interest (ROI) and computes metrics that are afterward studied to quantify the patient's seizures.

For each ROI, different MOI can occur. Using the Graphical User Interface (GUI), the user can watch the seizure video for identifying MOI and create them, followed by their labeling (classification, type, sub-type, clinical start and clinical end, for the specific seizure) that can be edit at any time.

For the tracking part, the user first chooses the ROI (e.g. head, left arm) and is then asked to select the area that delimits it, for the algorithm to track successfully. After completion, for all MOI, the maximum, minimum, median, mean and standard deviation (std) is calculated for velocity, acceleration and jerk (second derivative of velocity). Furthermore, the algorithm computes the movement displacement (MD), covered distance (CD) and movement extent (ME) for each ROI, besides the velocity, acceleration and jerk. All of these metrics, that can be analyzed in [Figure 2.2](#), are decisive for quantifying the patient's movements during a recorded seizure.

Sz Type	#MOIs	Velocity ( $\text{ms}^{-1}$ )		Acceleration ( $\text{ms}^{-2}$ )		Jerk ( $\text{ms}^{-3}$ )		MD (m)	CD (m)	ME ( $\text{m}^3$ )	MOI duration (s)
		Median	Sth	Mean	Sth	Mean	Sth				
TLE	19	0.086 ± 0.064	0.58 ± 0.41	12 ± 8.4	38 ± 33	345 ± 287	1204 ± 1151	0.15 ± 0.11	6.3 ± 4.2	0.068 ± 0.11	23 ± 6.6
ETE	23	0.068 ± 0.063	0.46 ± 0.29	8.6 ± 4.6	23 ± 17	172 ± 127	574 ± 506	0.44 ± 0.29	7.6 ± 2.9	0.14 ± 0.16	35 ± 10
$\rho$	n.s.	n.s.	n.s.	n.s.	<0.05	n.s.	<0.001	n.s.	<0.05	<0.01	

[Note.] Values are given as mean ± std. n.a.: not applicable. n.s.: not significant.

Figure 2.2: MOI parameters with highest  $\rho$ -score from patient's 42 seizures MOI. Adapted from [8]

## 2.4 Problem Description

Bearing in mind that the KiSA software is developed in Matlab, it has some issues associated with efficiency, cost (expensive license) and user-friendliness. The latter is mostly due to the current interface for KiSA, it has an unappealing look and is not intuitive, as it is presented on [Figure 2.3](#). But this problem can not have much improvement due to the limitation associated with using a GUI made with Matlab.

Moreover, the algorithm also has some complications: the tracking lacks accuracy in some cases, it has prolonged loading time, due to some heavy files being computed



Figure 2.3: KiSA's graphics user interface. Adapted from KiSA - Kinematics Seizure Analysis, Users' Guide v3

in the complex algorithm's logic, and some tasks can not be redone, *i.e.*, if the markings are applied poorly, the algorithm does not have the ability to retrack. All of these problems are also bounded by the programming language.

## 2.5 Requirements Outline

Due to the problems mentioned in the section above, a new version of the software is being developed - KiSA v3.0. Now the programming language used is Python, this makes it possible to add more features and the complications related to Matlab can be solved. Most noticeably, it becomes easier to solve the problem of the User Interface (UI) and User Experience (UX), being that Matlab's GUI will not be used anymore.

Given that the GUI for the new software is the most underdeveloped part, that is what this study will focus on. This thesis will aim to develop a fully functional interface, that can handle the different tasks created by the user's interaction. And considering that the new version of the algorithm will be incorporated in future iterations past this thesis, the code developed in this study needs to be flexible and maintainable for any programmer to understand and continue the KiSA improvement.

Furthermore, it is desirable to run this application on a web server. To do so, alongside with the front-end development, it is also necessary to create a way for the back-end to communicate with the user side in the most time-effective way available. The main goal here is to decrease the user's waiting time as much as possible.

All that being said, when choosing the framework and tools to be used on this project, it needs to be taken into consideration the following requirements:

- Fast response time
- Efficient communication between back-end and front-end
- User-friendliness
- Usability
- Flexibility of adding new features



# Chapter 3

## State of the Art

This chapter cover a series of studies made to determine the best solution for the problem exposed previously, keeping in mind the requirements needed.

### 3.1 Concept Review

To better understand the studies made, it is important to define some concepts that are the base for the technologies shown in this chapter.

#### 3.1.1 Framework Definition

First and foremost, it is important to underline the difference between a framework, a Toolkit and a library.

A Toolkit is a group of tools that are used to aid with the access of a system. A Library is a code file that can be invoked from another one, to assist with the development of a new code.

On the contrary, frameworks are platforms that create applications in a standard way, offering more than one service, as most libraries and Toolkits do [9].

To develop a solution, it will be necessary to use some of these tools. This because, if a project was developed purely with its core language, it would require an exhaustive knowledge and workload for the programmer and, as a consequence, the results would take longer to achieve [10]. It is not necessary to waste time developing a concept that is already established and optimized.

#### 3.1.2 HTTP Methods

Since there is a possibility for this work to be developed as a web application, it is best to understand first how an internet page can communicate with a server<sup>1</sup>, where the necessary data will be stored.

A web page can communicate with a server using Hypertext Transfer Protocol (HTTP), a protocol created for this purpose. When the client makes a request, HTTP will send a request message to the server for it to respond with the necessary information. Some of the available methods are GET, POST, PUT, DELETE, etc. [11].

---

<sup>1</sup>Software or computer that provides services and functionalities to other programs or devices

For better organization and security, each method should be used only for a specific condition [12]. Next are some examples of these HTTP methods, the ones most relevant for this thesis, and their use constrains:

- GET - retrieve data from the server
- POST - submit information to the specified resource
- PUT - replace data already present in the server
- DELETE - eliminate a specified resource

### 3.1.3 API

Once a project goes to a browser page, it is necessary to make use of a web API to handle these HTTP requests. To better understand this architecture we can resort to [Figure 3.1](#), where it is demonstrated how a user can make a request via the browser and the API handles the data exchange with the application.

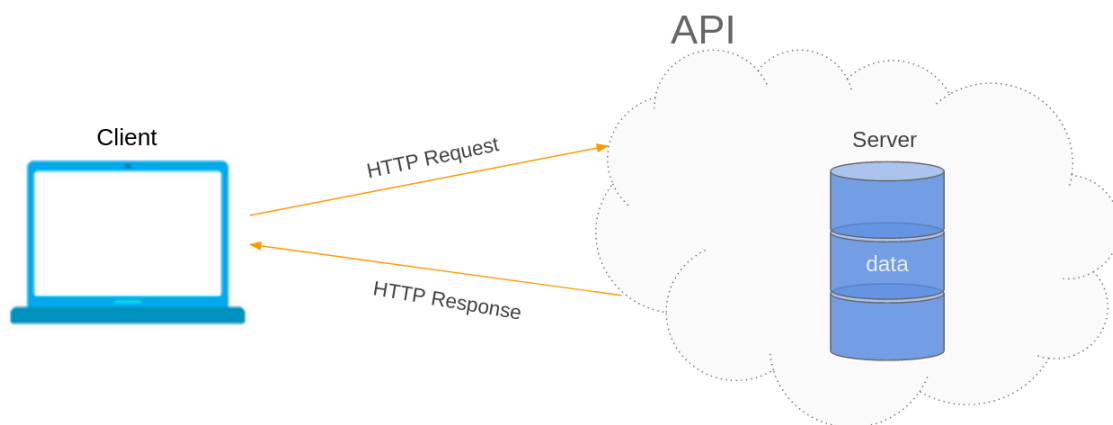


Figure 3.1: API architecture with HTTP requests

#### 3.1.3.1 Postman

With the purpose of helping build and test the correct behavior of the API, this study will make use of Postman<sup>2</sup>, a software development tool that enables the developer to call its API and review the returned information for different types of requests [13].

Postman is packed with multiple features that allow us to perform almost any HTTP Request. Furthermore, it allows for API testing by writing a script for the platform to validate, in order to ensure that everything is working as expected

This option will bring many advantages such as time optimization, being that the software handles all the necessary requirements regarding the client-side. This way, problems like browser compatibility, such as CORS<sup>3</sup>, will not be a problem when trying to optimize and debug the API.

<sup>2</sup><https://www.postman.com/>

<sup>3</sup><https://developer.mozilla.org/pt-PT/docs/Web/HTTP/CORS>



## 3.2 Front-end Development

Previously, KiSA was being developed as a native application, meaning that it is designed for a specific platform, desktop in this case, it does not run on a web browser.

Building a native application can bring some advantages, considering that the programmer does not need to worry about browser behavior and compatibility. They provide optimized performance and take advantage of the latest technology, due to the ability to use device-specific hardware and software [14]. However, a web interface can offer more resources, due to the massive support that browsers give, unlike native GUIs that are limited to their own assets.

That said, it is important to study both options and conclude which one is better to solve the issues at hand. Hence, a group of the most popular platform-independent GUI toolkits and web libraries (and frameworks) were studied in the sections below.

### 3.2.1 Native GUI Toolkits

It is important to stress that the research for this section was made exclusively in Python because the KiSA algorithm is being developed in that language. Therefore it is easier to integrate and it will bring fewer complications in the future due to lack of compatibility. Furthermore, it becomes more convenient and simpler to add new features.

Nowadays, Python has an enormous amount of GUI frameworks and toolkits that are used to develop desktop GUIs, being platform-specific (*a.k.a.* native) technologies or cross-platform solutions. Some of the toolkits employed are Tkinter, wxPython, Kivy and Qt (via PyQt or PySide) [15].

#### 3.2.1.1 Tkinter

Tkinter is a toolkit used to developing GUIs for multiple applications. It can be compatible with Linux, MAC OS and Windows.

Along with having a large number of examples and tutorials available, such as the one where [Figure 3.2](#) is from, Tkinter also has additional modules, widgets, geometry management and event handling to assist with the developed work [16]. This can result in a very convenient and compatible programming experience.

One of the downsides of using this solution is that Tkinter does not have an automatic GUI generator, *i.e.*, while coding the developer can not visualize real-time graphics design and layout [17].

#### 3.2.1.2 PyQt

PyQt is another toolkit to develop GUIs for multiple applications. It can run on Windows, OS X, Linux, iOS and Android.

This option is available in both PyQt4 and PyQt5 edition. It contains over 600 classes, these include tools for creating GUI, XML handling, network communication, SQL databases, Web browsing and other technologies available in Qt, such as



Figure 3.2: Python GUI Programming (Tkinter). Adapted from [18]

abstractions of network sockets, threads, regular expressions, SVG, OpenGL, XML, among others [19, 20].

Every element present in the interface is a widget. From buttons to labels, windows to dialog, progress bars, etc., as it can be seen in Figure 3.3. Another practical thing about the interface is that the developer can choose whether to create it by coding or using Qt Designer (Figure 3.4).

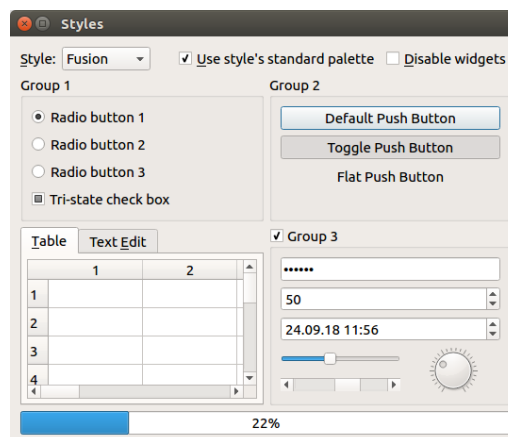


Figure 3.3: a GUI with Python and Qt, using “Fusion” style. Adapted from [21]

### 3.2.1.3 WxPython

WxPython is an open source<sup>4</sup> toolkit for developing the application’s GUI. Projects made with it have a native appearance on all platforms, being it Windows, Mac and Linux or other Unix-like systems [23].

The biggest difference from the solutions mentions before, it that this one does not come included with Python, meaning that it requires additional installation.

<sup>4</sup>[https://handwiki.org/wiki/Open\\_source](https://handwiki.org/wiki/Open_source)

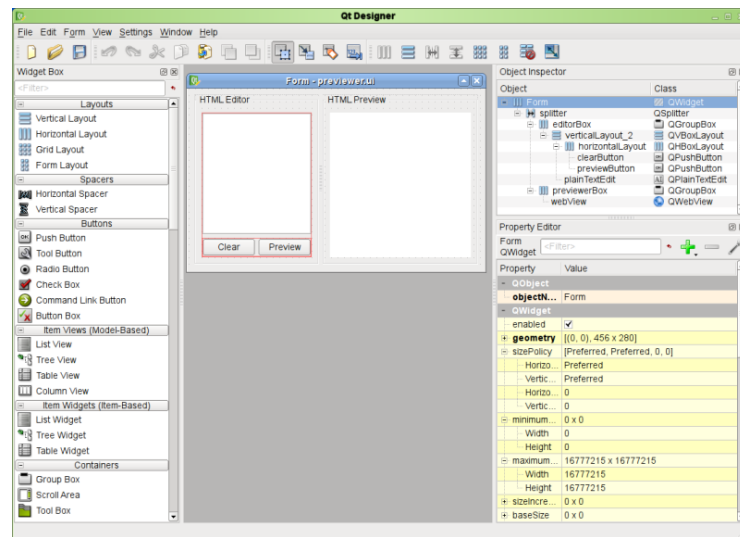


Figure 3.4: Qt Designer. Adapted from [22]

It is also the toolkit that has the least amount of documentation, compared to the previous cases. So it can become very difficult to develop an application with this set of tools.

Regarding the visual interface, it has a very minimalist look (Figure 3.5). But there is a wide range of widgets available.

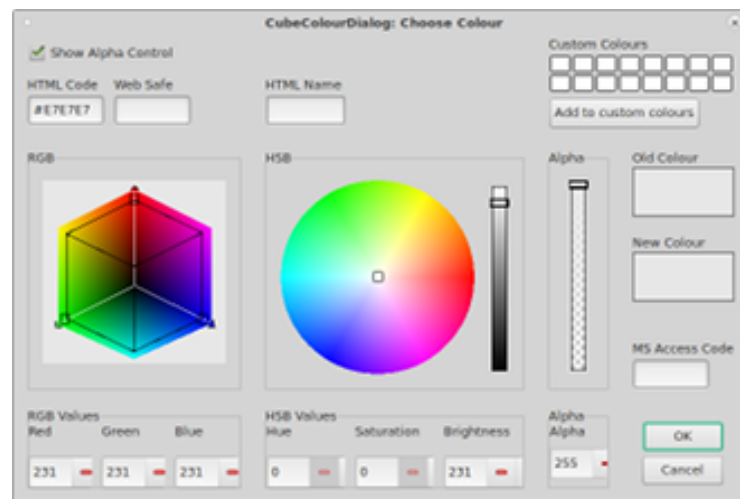


Figure 3.5: CubeColourDialog, wxGTK Control Appearance. Adapted from [24]

### 3.2.1.4 Kivy

Kivy<sup>5</sup> is an open source library that is mainly used to create applications that have multi-touch interfaces, such as mobile apps<sup>6</sup>. It is compatible with iOS, Android,

<sup>5</sup><https://kivy.org/>

<sup>6</sup>Software designed to run on a mobile device

Raspberry Pi, Linux, Windows and MacOS. It provides support for networking protocols and remote login, in addition to also being published under the MIT license.

This library comes with a multiple set of widgets that are used in the development of a GUI, but only a basic amount is offered, even though they are all highly extensible [25].

Being open source, it is developed by a community and is free to use, but there's a possibility that some questions regarding the library are left unanswered. Currently, about one-third of them are left ignored.

In terms of appearance, Kivy has its own GUI style, as displayed in Figure 3.6. It is based on OpenGL, as such, it uses a modern and fast graphics pipeline.

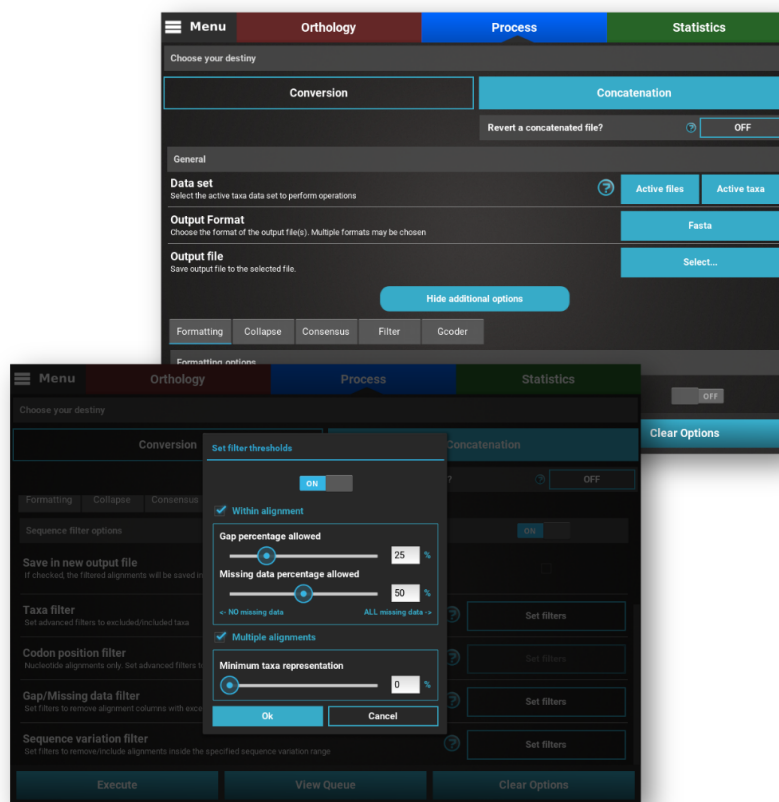


Figure 3.6: Graphical interface of “TriFusion”. An application made using Kivy

The main purpose of this library is to create an easy and fast interaction, as well as making the developed code reusable and deployable. In fact, this library tries to abstract basic tasks and this can make it both easy to use and easy to extend [26, 27].

### 3.2.2 Web Libraries

Instead of having a native application, it is possible to run the project on a web browser. This gives some advantages as to the previous solution, such as access to the application simply by having an internet connection, not being necessary to

install the software on the user's platform. Also, the range of choices and resources when building the interface is much larger than with a native GUI.

To build a browser application, it is possible to apply either Vanilla<sup>7</sup> HTML, CSS and JavaScript, or to make use of a library or a framework. As explained in [Subsection 3.1.1](#), it is preferable to use the latter due to time optimization.

In this subsection, there will be studied a group of the most used frameworks and libraries to develop web applications.

### 3.2.2.1 React

React is an open-source JavaScript library developed by Facebook<sup>8</sup> and used not only by the company but also in its own products, including Instagram<sup>9</sup> and WhatsApp<sup>10</sup>.

It was developed with the purpose of creating interactive, stateful, and reusable interface components. Therefore, compared to other solutions mentioned in this section, React offers a lighter option because it is filled with fewer conditions and does not need to use extra elements [28]. Additionally, given that it is a library, it becomes easier to integrate with other pieces of code, making this option a more flexible choice.

This technology can be easily learned, giving that it has the most amount of activity on GitHub<sup>11</sup>, as it is shown on ([Figure 3.7](#)) the assistance is vastly superior to the other options.

However, applications developed in React often need additional libraries to optimize the processes of state management, routing, and interaction with the API [29].

Furthermore, most of the components used in React are developed by the community. This can create some problems, giving that features may be lacking or insufficient, nevertheless, it is possible that tutorials and references are available online.

Moreover, it is important to add that React Native is a framework specifically design for a native build, more specifically mobile applications [30], and developed using React. So, if in the future there is a desire to turn this project into an Android or iOS application, it is very easy to convert the developed code into the needed software.

### 3.2.2.2 AngularJS

Angular is an open source JavaScript framework, that is maintained by Google and by the vast community of contributing programmers.

Even though it is one of the most popular frameworks for web development, the biggest disadvantage of this solution is its steeper learning curve ([Figure 3.8](#)). Given its large volume, knowing all the concepts is more time consuming than with React and, the fact that it is a framework, involves a complicated set of syntax errors, that will contribute to the increasing of the learning time [31]. Nevertheless, this solution comes with detailed documentation that can improve the development time.

---

<sup>7</sup>[https://handwiki.org/wiki/Vanilla\\_software](https://handwiki.org/wiki/Vanilla_software)

<sup>8</sup><https://www.facebook.com/>

<sup>9</sup><https://www.instagram.com/>

<sup>10</sup><https://www.whatsapp.com/>

<sup>11</sup><https://github.com/>

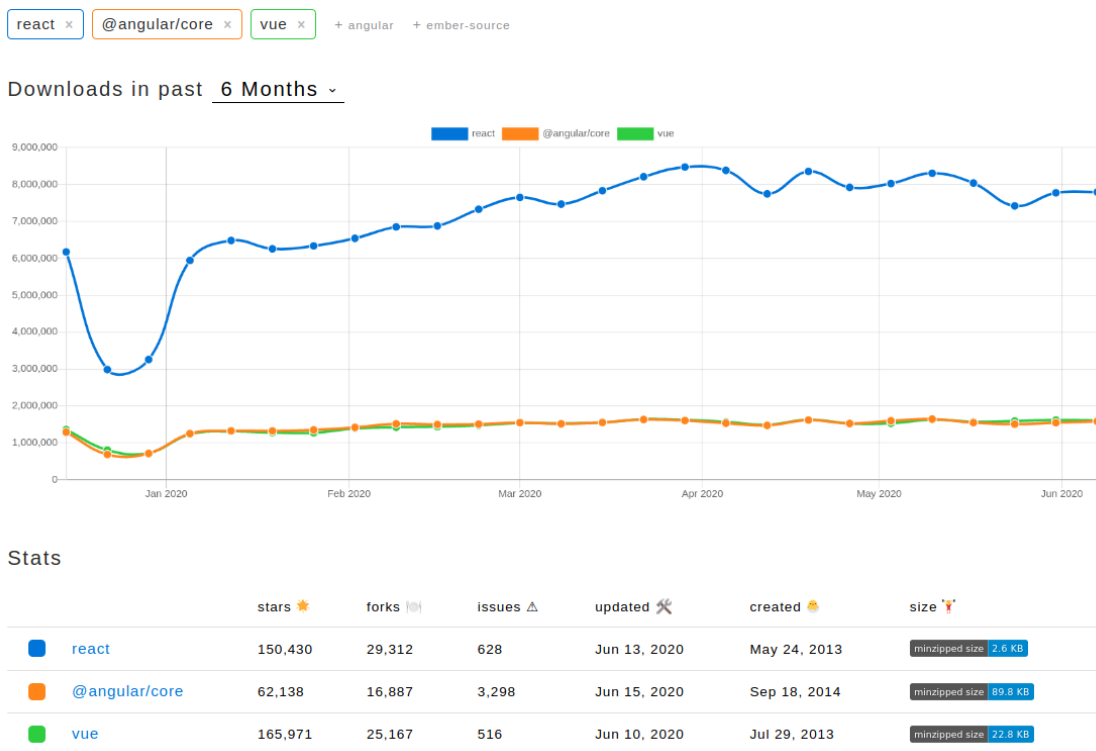


Figure 3.7: GitHub Stats: React vs. Vue vs. AngularJS. Data acquired from [www.npmtrends.com](http://www.npmtrends.com) on July 2020

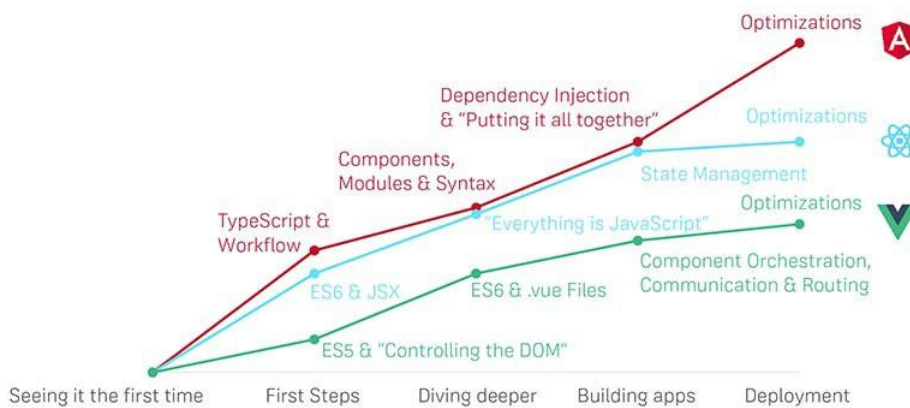


Figure 3.8: (Possible) Learning Curve for AngularJS, React and Vue. Adapted from [32]

### 3.2.2.3 Vue.js

Vue.js is another JavaScript framework. It is also used for developing user interfaces but mostly for single-page applications.

Vue.js does not have an abrupt learning curve, as seen on [Figure 3.8](#), only requiring programmers to know the basics of JavaScript, HTML and CSS, which is unlike Angular or React. The idea behind the development of Vue.js is to achieve good results with as little effort as possible, making the code complete with just a few lines [33].

It is very lightweight and is composed of a simple structure, so it is easy to develop large and reusable templates. It also has a marvelous integration capability, making it useful to build both single-page projects and complex web applications.

Despite its lack of resources, the extensive and detailed documentation can assist with many problems. Nevertheless, the amount of popularity with this framework, compared to the previous ones, is very low as shown in [Figure 3.7](#).

### 3.2.3 Conclusion

Amongst the studies shown above relative to native GUIs, some conclusions can be made.

Tkinter seems to be the least viable option, having a limited amount of widgets available. Also, its outdated visual does not make a strong candidate to resolve the problem at hand.

PyQt is a very compelling choice, given that it has multiple tutorials and third-party projects available to study from, alongside with having an active community to solve any doubts that may arise.

WxPython has a plus of having a native appearance in every platform, giving more uniformity to the group of developers. But the lack of resources to search from, compared to other more viable solutions, makes this one a hesitant choice.

Together with the fact that Kivy is the solution that has a more distinctive look, having also an MIT license makes it a very reliable and appealing library.

But even though most of these options seem viable, making a native GUI is still very constricted. Having several limitations in each of its components, some ideas for the interface might not be conceivable due to the lack of resources available. That being said, the next option can simplify the problem at hand and ease the development process, giving extra time to conceive more ideas that are attached to this project.

Regarding web frameworks, React and AngularJS seems to be the most obvious choices, being that they are the ones used in big companies, such as Facebook, WhatsApp, Google, among many other, and have the most amount of documentation and community support. Even though Vue.js is very easy to learn, its smaller assistance makes it a less viable option, either by the lack of resources or the small amount of answered questions available, even though its documentation is quite detailed.

At first glance, Angular has more features than React but takes more time to learn, given that its structure is fixed and complex, more suitable for experienced developers [29].

React has a more steep learning curve than Angular, less time to learn means more time to develop the product. Nevertheless, the amount of resources that React has can sometimes be overwhelming, due to the number of documentation and information available online, not knowing which one is the best or if it is all reliable and useful. But, in the long run, this library is the better option considering that, after the basics established, it becomes very easy to develop the application desired.

Additionally, if in the future there is a desire to extend the application as a mobile app, it's possible to use React Native. It is a library also developed by React, indicated for mobile development, with very similar language. Bearing that in mind, making the conversion to mobile, if necessary, will not be such a big of a challenge.

In conclusion, the best approach to handle the interface creation is to run the application on a web server, with React as its source of front-end development.

In this regard, it is now necessary to find a way of sending the data from the browser to a local server, where the information acquired from the user's interaction will be stored. That server will run in a virtual machine available at INESC-TEC's laboratories.

As such, to develop the communication between the front-end and the back-end, it is imperative to study a group of services that will perform this communication as quickly and effectively as possible, as it is a requirement mentioned in [Section 2.3](#).

### 3.3 Web API Development

Even though KiSA was previously developed as a desktop application, the goal now is to run it on a web browser. With this in mind, to create the API a group of web development frameworks is studied in the subsections below, to conclude the best alternative to develop said requirement.

#### 3.3.1 Web Frameworks

Once again, the research for this chapter was also made exclusively in Python, for the same reasons stated on [Subsection 3.2.1](#).

Nowadays, Python is used in many application domains. It can be used for GUI development, as mentioned above, along with web development, using frameworks such as Django and Pyramid, or micro-frameworks like Bottle and Flask. Furthermore, Python's standard library supports many Internet protocols namely HTML, XML, JSON, among others.

Amongst the frameworks available, Django and Flask are the ones to choose from each category, being that they are the most popular amongst the options presented. They are both open source, which means there is an active community maintaining them both. As for other Python frameworks, they simply do not come close to Flask and Django, based on the number of contributors, forks and stars from each framework's GitHub, as shown in [Figure 3.9](#) and [Figure 3.10](#)



django/django		Pylons/pyramid	
★ Stars	49,992	★ Stars	3,383
🔗 Forks	21,697	🔗 Forks	867
📄 Open issues	222	📄 Open issues	62
📅 Age	8 years ago	📅 Age	10 years ago
➕ Last commit	14 hours ago	➕ Last commit	12 days ago
⚙️ License	NOASSERTION	⚙️ License	NOASSERTION
🗨️ Language	Python	🗨️ Language	Python
Remove repo		Remove repo	

Figure 3.9: API Frameworks GitHub Stats: Django vs. Pyramid. Data acquired from [www.githubcompare.com](http://www.githubcompare.com) on July 2020

bottlepy/bottle		pallets/flask	
★ Stars	6,845	★ Stars	50,790
🔗 Forks	1,319	🔗 Forks	13,561
📄 Open issues	285	📄 Open issues	27
📅 Age	11 years ago	📅 Age	10 years ago
➕ Last commit	14 days ago	➕ Last commit	14 hours ago
⚙️ License	MIT	⚙️ License	BSD-3-Clause
🗨️ Language	Python	🗨️ Language	Python
Remove repo		Remove repo	

Figure 3.10: API micro-frameworks GitHub Stats: Bottle vs. Flask. Data acquired from [www.githubcompare.com](http://www.githubcompare.com) on July 2020

### 3.3.1.1 Django

Django is a high-level<sup>12</sup> Python web framework that favors rapid development and clean code design. Being built by experienced programmers, it handles most of the complications of web development, leaving the user more time to work on their project. As mention before, it is also open source, therefore free to use. Examples of companies that make use of this technology are Youtube<sup>13</sup>, Spotify<sup>14</sup>, Dropbox<sup>15</sup>, among others.

<sup>12</sup>[https://handwiki.org/wiki/High-level\\_programming\\_language](https://handwiki.org/wiki/High-level_programming_language)

<sup>13</sup><https://www.youtube.com/>

<sup>14</sup><https://www.spotify.com/pt/>

<sup>15</sup><https://www.dropbox.com/>

Created for the quick development of database-driven sites, it is scalable, mature and has a robust set of built-in components [34, 35]. Moreover, its ability to generate most of the application structure automatically and its lightweight syntax minimizes the amount of code that needs to be written, as well as increases the software's response speed.

Furthermore, it has a layer of security that ensuring that developers do not commit any mistakes related to SQL injection, cross-site request forgery and cross-site scripting [36].

This framework uses a structure based on a Model-View-Controller (MVC) pattern. This means that the architecture of the code is divided into the following objects:

- **Model** - pure application data, containing no logic
- **View** - data presentation to the user. Here the application knows how to access data but not how to manipulate it
- **Controller** - events, triggers listeners and the reaction to them. This layer exists between the view and the model

MVC mostly relates to the interaction layer of an application. Even though it gives the advantage of parallel development, as each programmer can work simultaneously on any part, this can become complex to implement, due to the separate layers' complexity and the abstraction it requires. [37]

Another downside of this approach is the deep learning curve, making it unfit for small projects because of the large amount of software to grasp and the overwhelming number of features that it contains.

Nevertheless, it is one of the most popular frameworks nowadays, so it is not surprising that a wide variety of documentation and tutorials are available for public use. Furthermore, the large number of public projects allows for the possibility of reusing some components already implemented.

### 3.3.1.2 Flask

Flask is a micro-framework that has recently become more popular than Django, used in application such as LinkedIn<sup>16</sup>, Netflix<sup>17</sup> and Reddit<sup>18</sup>. It is also open source, has fully documented tutorials and an active community willing to offer support.

It was born due to an April fool's joke, giving that the premise was to create a framework with only a single file. Evolving to this day, it has become a useful tool for creating projects quickly [38].

This solution implements a bare-minimum web server, but without sacrificing power. That being said, Flask applications result in simple interfaces, therefore, this solution is primarily aimed at small projects with elementary requirements [39, 40]. However, this does not mean that large applications can not make use of this technology, because the provided blueprints can greatly simplify the workflow, which is Flask's concept of modularity.

---

<sup>16</sup><https://www.linkedin.com/>

<sup>17</sup><https://www.netflix.com/>

<sup>18</sup><https://www.reddit.com/>

Unlike Django, developers must write most of the code themselves, but the amount of lines written is very small. The file structure is rather elementary consisting of very few files, keeping the core simple, very extensible and flexible, characterizing this solution as a micro-framework.

However, the development process can become quite restricted, given the limited amount of features (compared to Django). Moreover, the inability of handling asynchronous tasks is also a disadvantage, but this problem can be remedied with the use of an external task queue, such as Celery<sup>19</sup>.

### 3.3.1.3 Conclusions

Django can be quite useful when building an MVC application, but back-end projects that need a simple web interface, fast to develop and require little configuration, often benefit from Flask [39].

If the final goal for building this API were to reuse as many resources as possible, Django would have fewer obstacles in its way. But given that this API will be used for a small part of this project, only being necessary to transfer small information, it is more appropriate to use Flask, as it requires very few lines of code to make simple HTTP requests.

Furthermore, choosing a micro-framework is more logical considering that very few HTTP methods are necessary, thus a quick and simple solution would be more suitable for the API development.

## 3.3.2 Data Formats

When the API receives data from the application it is necessary to store it in some database or file. To simplify the architecture, the data received from KiSA will be store on a file. This leads to the next problem: which type of file format should be used to store information?

There are many types of data data structures to choose from but, currently, the ones most used are JavaScript Object Notation (JSON), Extensible Markup Language (XML) and Comma-Separated Values (CSV), being the first two much more famous than the latter one for web APIs, as shown on [Figure 3.11](#).

### 3.3.2.1 JSON vs. XML

Even though both of these languages are very similar, XML requires a closing tag for each element. This makes JSON code easier to read and smaller. As we can see from [Listing 3.2](#) and [Listing 3.1](#), for the same the same example JSON requires fewer code lines.

---

<sup>19</sup><https://flask.palletsprojects.com/en/1.1.x/patterns/celery/>

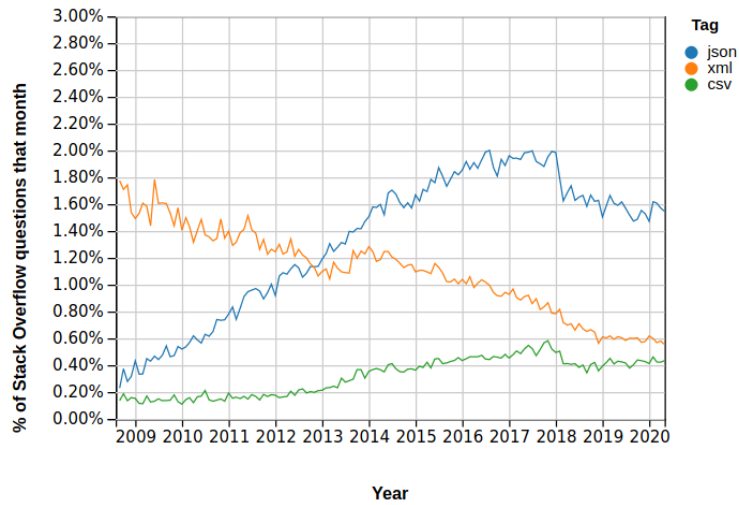


Figure 3.11: Stack Overflow Trends: JSON vs. XML vs. CSV. Data acquired from <https://insights.stackoverflow.com/trends> on July 2020

```

1 <employees>
2   <employee>
3     <firstName>John</firstName> <lastName>Doe</lastName>
4   </employee>
5   <employee>
6     <firstName>Anna</firstName> <lastName>Smith</lastName>
7   </employee>
8   <employee>
9     <firstName>Peter</firstName> <lastName>Jones</lastName>
10  </employee>
11 </employees>

```

Listing 3.1: XML Example Code. Adapted from [41]

```

1 {
2   "employees": [
3     { "firstName": "John", "lastName": "Doe" },
4     { "firstName": "Anna", "lastName": "Smith" },
5     { "firstName": "Peter", "lastName": "Jones" }
6   ]
7 }

```

Listing 3.2: JSON Example Code. Adapted from [41]

For this reason, JSON is the obvious choice to store data from the API, as it is the option quicker to read and write and generally the encoding of JSON is faster than using XML [42].

## 3.4 Desktop Applications using Web Technologies

As mentioned before, developing a native GUI is very limited due to the immutable widgets offered by the toolkits available especially in the video components, which will be one of the main and most important features of the interface. As a result, developing a web interface is much more flexible because almost anything can be done with vanilla HTML, CSS and JavaScript [43].

Nevertheless, this does not mean that it is impossible to develop a desktop application. Bearing in mind that this project will be programmed for a web browser, it is possible to convert the work done into a native application, using the tools described in this section. That way we can have two different benefits at once, take advantage of web frameworks' flexibility and native applications' compatibility.

### 3.4.1 Electron

Electron is an open source framework for creating native applications with web technologies like JavaScript, HTML, and CSS

It is one of the most famous, if not the most popular cross-platform desktop development tool. Countless applications were developed using this framework, as can be seen on Electron's official website<sup>20</sup>, with examples like WhatsApp<sup>21</sup>, Mattermost<sup>22</sup>, Discord<sup>23</sup>, Skype<sup>24</sup>, Slack<sup>25</sup>, Twitch<sup>26</sup>, among many others. Furthermore, the design remains exactly the same for both browser and desktop, and across any operative system.

Even though there are some other options that can be better than Electron [44], most of them are paid. Furthermore, the lack of known projects developed with other frameworks, compared to Electron is ludicrous.

### 3.4.2 Proton Native

Another possible solution that is worth mentioning is Proton Native. It is a cross-platform desktop development tool, with a React environment, developed as a small project, for desktop application building. Some examples of projects that make use of this software are.

This option can become a better solution than Electron due to bringing less overhead, using native tools with a smaller size and with less resource usage [45]. Furthermore, it has better system compatibility, it uses the same syntax and components as React Native, therefore it can become very easy to use, due to the original program being developed with a similar coding language.

---

<sup>20</sup><https://www.electronjs.org/apps>

<sup>21</sup><https://www.whatsapp.com/>

<sup>22</sup><https://mattermost.com/>

<sup>23</sup><https://discord.com/>

<sup>24</sup><https://www.skype.com/pt/>

<sup>25</sup><https://slack.com/intl/pt-pt/>

<sup>26</sup><https://www.twitch.tv/>

electron/electron	kusti8/proton-native		
★ Stars	83,457	★ Stars	10,337
🔗 Forks	11,204	🔗 Forks	351
📄 Open issues	1,243	📄 Open issues	32
📅 Age	7 years ago	📅 Age	3 years ago
➕ Last commit	14 hours ago	➕ Last commit	18 days ago
🛡 License	MIT	🛡 License	MIT
🗣 Language	C++	🗣 Language	TypeScript
Remove repo		Remove repo	

Figure 3.12: GitHub Stats: Electron vs. Proton Native. Data acquired from [www.githubcompare.com](http://www.githubcompare.com) on July 2020

### 3.4.3 Conclusion

Due to time constraints, it would not make sense to develop the desktop application, because this period would be best spent trying to improve any poor details or to add more features to the main web solution.

Furthermore, a desktop application gives fewer advantages than a web one. The software could be too heavy to run on a weaker computer and it would be a requirement to install the application on each machine, as well as manually update every time the software changes. Compared to the simplicity of opening a web page, only being necessary to have an internet connection, this option is much preferable than the previous one.

However, a native development can always be done as future work, if necessary, as the frameworks described in this section show, this option would not be very hard to develop.

## 3.5 Related Projects

To assist in the development of these projects, it is important to study other applications related to EEG analysis and epilepsy assessment, to try and develop the best solution possible for the final user (neurologists and experts with neurological diseases) to be pleased with the results. This way, it becomes easy to come up with ideas for the interface and maybe get some inspiration in the front-end development.

In order to gather the best possible projects available, this study resorted to the Epilepsy Foundation<sup>27</sup>, the leading non-profit US organization devoted to helping treat epilepsy. More specifically, the 2020 Epilepsy Pipeline Conference was investigated, to collect the project's studies in this section [46]. This conference showcases the latest developments related to epilepsy, regarding both technology and drugs but

<sup>27</sup><https://www.epilepsy.com/>

this work will obviously focus on the former, more specifically in the Diagnosis and Detection Device Presentations.

Most of the technologies presented, mainly make use of some kind of wearable, such as Epilog<sup>28</sup>, Embrace 2<sup>29</sup>, 24/7 EEG SubQ<sup>30</sup> among others. And do not have video analysis, contrary to KiSA. The project that most resembles the current software is the one from Neuro Event Labs and Epihunter, which will be further studies below.

There are other software and tools outside the 2020 Epilepsy Pipeline Conference, but most of them rely on EEG examination [47] and are not as technologically advanced as the solutions mentioned.

### 3.5.1 Neuro Event Labs

Neuro Event Labs is a company that develops solutions for registering seizures on epilepsy patients. Similar to NeuroKinect (Section 2.2), a camera is present on the scene to collect the video data. Additionally, they also have other audio and movement sensors to detect changes in the patient's heart rate or breathing [48], the NeuroKinect does not make use of this last sensors.

This company makes use of an intelligent algorithm, Nelli [49], that combines computer vision and machine learning techniques to analyze video data and compute quantitative measurements for human movement. Nevertheless, even though the video approach is very similar to NeuroKinect, it does not qualify each individual MOI.

The interface for Nelli is shown on Figure 3.13. As we can see, alongside the information calculated, the analysis also contains a video portion for the user to visualize alongside with all the metrics.

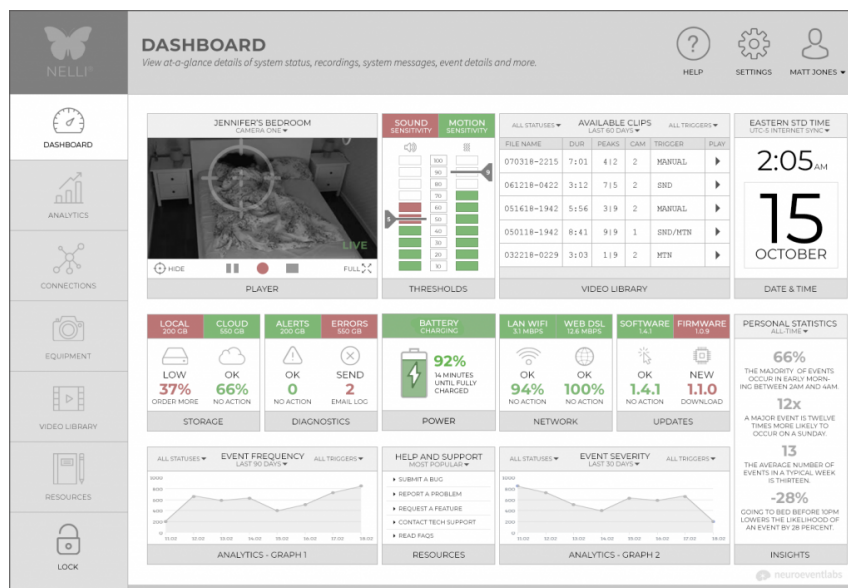


Figure 3.13: Nelli Initial Dashboard Design. Adapted from [50]

<sup>28</sup><https://www.epitel.com/>

<sup>29</sup><https://www.empatica.com/en-int/embrace2/>

<sup>30</sup><https://www.uneeg.com/en/products/24-7eeeg>

### 3.5.2 Epihunter

Even though this solution makes use of wearable devices, it is one of the few that also makes use of video analysis.

Epihunter is a Belgium digital health-tech startup that developed a solution to better monitor epileptic episodes at home. This technology detects, logs and signals silent absence seizures by measuring and recording brainwaves, using a wearable EEG headset, BrainLink Lite [51]. This company also developed a complementary app, Epihunter Companion seen on [Figure 3.14](#), to record the output of the important events on smartphones.

The software detects seizure activity and registers it in a logged diary, as well as plotting EEG samples. Alongside with it, a video can be automatically recorded 30 seconds before and after the detected activity or logging [52].

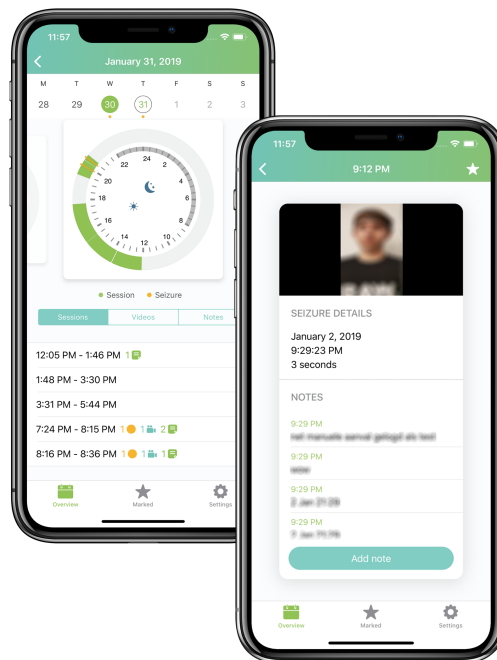


Figure 3.14: Epihunter Companion app. Adapted from [53]



## Chapter 4

# KiSA v3.0 Design and Implementation

### 4.1 Methodology

At the beginning of this study, in order to guarantee that the interface was appealing and intuitive enough, a group of initial drafts for the front-end were created, *a.k.a.* Mockups. They have the intention of giving a visual representation of the final application's front-end and improve the user experience when presented with the project's interface.

These Mockups have suffered some alterations over time, whenever new ideas emerged or a better organizational structure was suggested. When finished, the results were presented to a neurology specialist doctor from Hospital São João (HSJ)<sup>1</sup>, that agreed with the product developed so far and encouraged its progress.

After, this work adopted an iterative process where the code was developed and, after an iteration was ready, the supervisor and some of the BRAIN members would view the final result for approval and constructive criticism. As such, this project had a continuous development with constant feedback, to achieve the best solution possible while keeping all the members involved on the same page.

Similar to the Mockups approval, the interface was well accepted in its final iteration, by both the BRAIN members involved and the already mentioned specialist, that helped in the evolution of this project.

To fulfill the requirements mentioned on [Section 2.5](#), a group of Use Cases was established in order to list the necessary features that the system must have. Some of these items were already specified in the previous version of KiSA, but most of them are related to the interface improvement, to give the user a better understanding of the system's traits.

#### 4.1.1 Use Cases

This subsection describes the features that KiSA v3.0 interface possesses, by listing every functionality of the project, from the user's point of view, and capturing the way

---

<sup>1</sup><https://portal-chsj.min-saude.pt/pages/2>

that the system can be used. Use Cases characterizes how users will perform certain tasks on our website, outlining the system behavior as it responds to an action.

Some of these features are associated with the doctors' needs, therefore they were already present in the old KiSA version. However, the majority of them are associated with the old interface improvement, thus their development is essential in providing a superior UI and, overall, a better user experience. To better discuss the system's behavior, the following points explain each of the use cases developed.

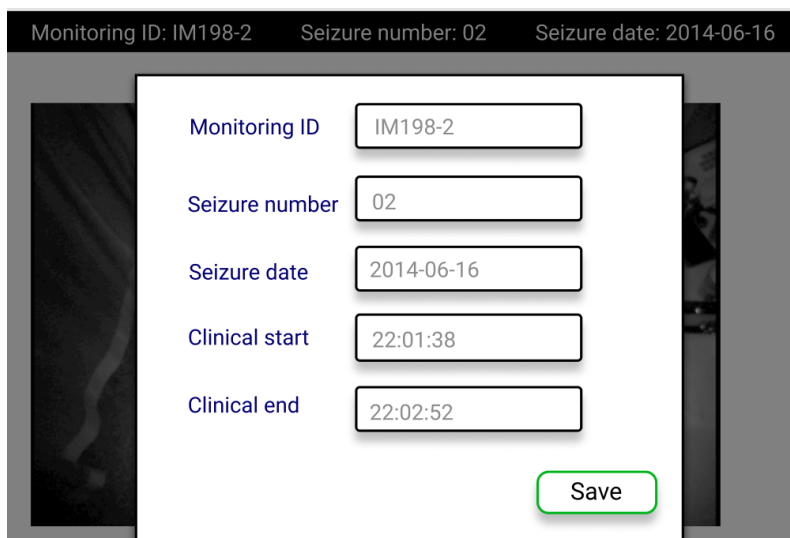
The images that will be shown in this section are extracted from the Mockups that were developed initially, using a prototyping tool called Figma<sup>2</sup>.

#### 4.1.1.1 Use Case 1: Pick up the session/exam

The doctor can study the patient progress or state at any point of the day, not needing the patient to be available for a medical appointment or even present in the hospital.

#### 4.1.1.2 Use Case 2: Edit the patient data

After selecting the correct file to analyze, the doctor has to fill in the patient data. This information must be shown at the top of the next page and the user should be able to edit it at any time when on this same screen, by just selecting the edit patient tab. This data will be necessary for the API, as it will be further described on [Chapter 5](#).



Monitoring ID: IM198-2    Seizure number: 02    Seizure date: 2014-06-16

Monitoring ID	<input type="text" value="IM198-2"/>
Seizure number	<input type="text" value="02"/>
Seizure date	<input type="text" value="2014-06-16"/>
Clinical start	<input type="text" value="22:01:38"/>
Clinical end	<input type="text" value="22:02:52"/>

Figure 4.1: Use Case 2 exemplification

<sup>2</sup><https://www.figma.com/ui-design-tool/>

#### 4.1.1.3 Use Case 3: Distinguish between body parts that have already been tracked for ROI

Alongside with a list of the available ROIs, a red background color should be associated with the ones that have not been tracked, and the ones that were already tracked should have a green background.

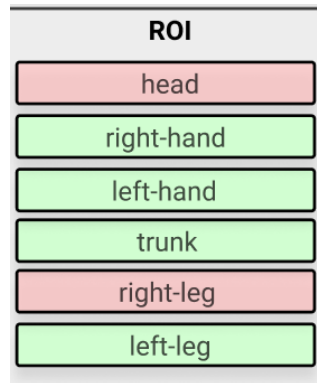


Figure 4.2: Use Case 3 exemplification

#### 4.1.1.4 Use Case 4: Highlight which ROI and which MOI is selected

Since there are multiple ROI and MOI, it is important to distinguish between the ones that are selected in order to have a better perception of what it is being done.

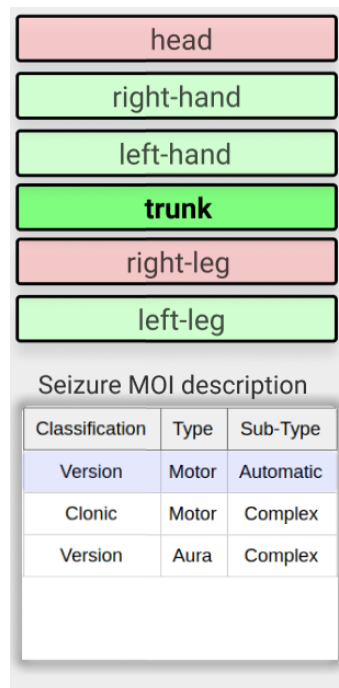


Figure 4.3: Use Case 4 exemplification

#### 4.1.1.5 Use Case 5: Tracking ROI

Using the KiSA new algorithm, the doctor should be able to track the ROI with the intention of computing the metrics needed for human motion assessment, as already explained on Chapter 2. These metrics can then be analyzed in the next screen. As it was not the case in the previous KiSA version, the user can now retrack these Regions of Interest, if they feel like the algorithm has not tracked the body part properly.

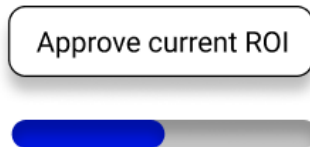


Figure 4.4: Use Case 5 exemplification

This tracking process can last for a little while. With the purpose of indicating the user that the system has not stopped working, a load bar should be implemented in this process.

#### 4.1.1.6 Use Case 6: Add and modify MOI labels

After selecting an ROI, the doctor should be able to add an MOI to that region and associate labels to it, with the purpose of adding more information to the analysis part. Furthermore, the user should be able to edit these labels at any point, if they wish to modify the work previously done.

Classification	Type	Sub-Type
Version	Motor	Automatic
Clonic	Motor	Complex
Version	Aura	Complex

Buttons: Delete MOI, Add MOI

Clinical Begin: 00:00  On current timestamp

Clinical End: 00:00

Classification: Version

Type: Motor

Sub-Type: Automatic

Save MOI

Clinical Begin: 08:44  On current timestamp

Clinical End: 09:10

Classification: Version

Type: Motor

Sub-Type: Automatic

Save MOI

(a) MOI table
(b) add MOI section
(c) MOI modification section

Figure 4.5: Use Case 6 exemplification

#### 4.1.1.7 Use Case 7: Manually choose clinical begin and end

For the MOI labeling, it is necessary to provide a clinical begin time and clinical end time. For this, the doctor should be able to enter the time manually, writing the numbers in the correct box.

Figure 4.6: Use Case 7 exemplification

#### 4.1.1.8 Use Case 8: Automatically select clinical begin and end

Parallel to the previous use case, for the time labels the doctor should also be able to pick them just by moving the video thumb and then selecting the radio box corresponding to the desired label. This way, the user can easily observe the patients' movements in the video and select the label based on the timestamp, not being necessary for them to introduce the time manually.

Alongside with this, it should be possible to skip the video frame by frame, thus having a very precise timestamp, not being necessary to drag the thumb to the exact millisecond.

Figure 4.7: Use Case 8 exemplification

#### 4.1.1.9 Use Case 9: Add seizure's classification, type and sub-type

Alongside the clinical begin and end, it is also necessary to provide three more labels. These include the Classification, Type and Sub-Type of seizure. They should be added when selecting the drop-down boxes available.

It is important to notice that the Sub-Type label depends on the Type. That being said, when certain kinds of Type labels are chosen, the next drop-down available should change its content, in order to adjust with the previous one chosen.

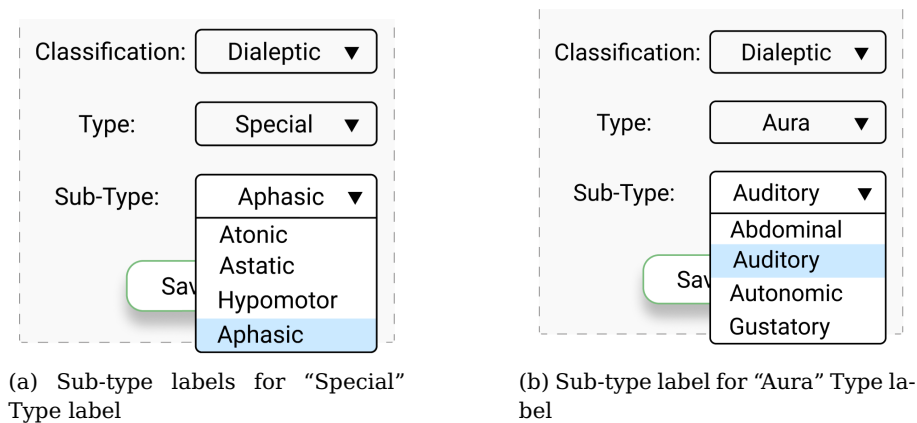


Figure 4.8: Use Case 9 exemplification

#### 4.1.1.10 Use Case 10: Visualize the raw video

The doctor should be able to view the video corresponding to the seizure selected. They should be able to perform the basic video controls: play, pause, stop, fast forward, wind back.



Figure 4.9: Use Case 10 exemplification

#### 4.1.1.11 Use Case 11: Highlight the MOI interval in the time track

When an MOI is selected, the time track should highlight the interval corresponding to that MOI period - from its clinical begin to its clinical end. This way, it is possible for the doctor to have a better time perception of the MOI duration.

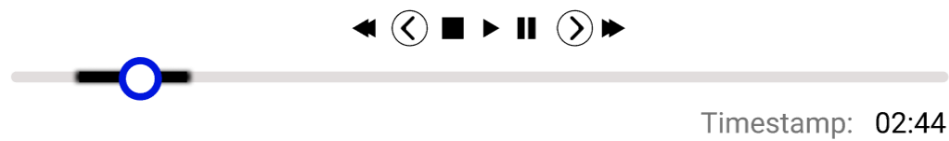


Figure 4.10: Use Case 11 exemplification

#### 4.1.1.12 Use Case 12: Play the video only in the selected MOI portion

It should also be possible to play the video only in this interval, by pressing the “Play MOI” button, after selecting a specific row from the table.

## 4.2 System Development

After the Mockups’ approval and between processes iteration, the KiSA interface was implemented using the code structure described below. All developed code was submitted to BRAIN’s GitLab<sup>3</sup> repository, created for this project’s progress.

Since the software is no longer being developed in Matlab, as mention in [Section 2.5](#), it gives the freedom to develop a better GUI. As such, we can build an interface that has a more intuitive and appealing visual aspect, as will be shown in this section.

As discussed on [Subsection 3.2.3](#) the tool most appropriate for this UI development is React. The way this library works is by using small and isolated pieces of code called components, to build a greater and compound page. In order to best organize the code, a component should be built as an independent unit and, when invoked, given attributes to correspond to the parent’s needs.

Therefore, a well-organized hierarchy is essential for a clean, easy to read, robust code that can be used in the future by other developers and increase the system’s scalability. In addition, this notion also contributes to the code quality and facilitates the debugging process.

### 4.2.1 Code Architecture

Every React project is composed by a *src* folder, where the main code will be developed. In this case, the directory not only has the *index.js* - where the code will start - but also the subfolders created for this project.

These directories are composed by the *screen* and the *components* folders. The former includes the different pages where the user can navigate to, and the latter is obviously for storing the components to be used as many times in each screen. Refer to [Figure 4.11](#) for a visual perception of the architecture. The *Screen.js* is where the page control happens. By using [BrowserRouter<sup>4</sup>](#) it is possible to render the correct screen, following the specific requirements. The *default* page is used for redirecting purposes at the beginning of the workflow.

<sup>3</sup><https://gitlab.com>

<sup>4</sup><https://reacttraining.com/react-router/web/api/BrowserRouter>

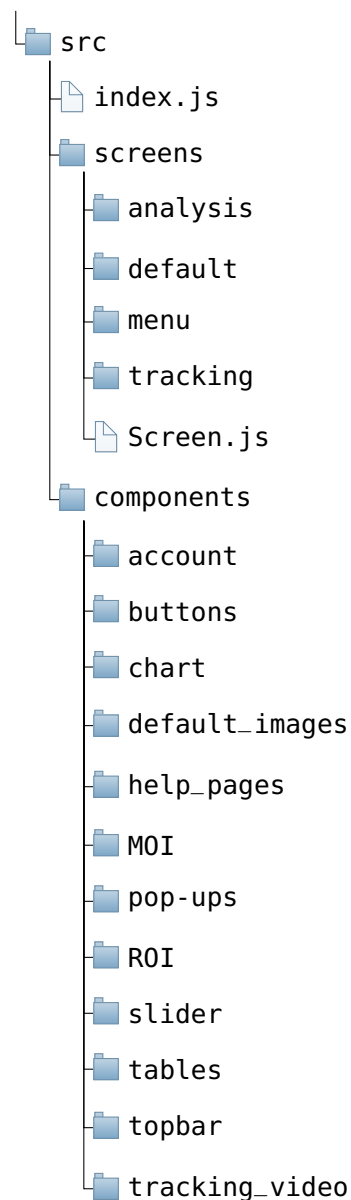


Figure 4.11: Code organization

The components can either be developed from scratch, or imported from a package. The former was chosen, given that the whole reason for using a framework is to optimize the development time and not dwell on things that were already created and function properly. That being said, most of the external components are imported from the *@material-ui/core*<sup>5</sup> package.

---

<sup>5</sup><https://material-ui.com/>



## 4.3 Results

The final interface is available in the following URL <https://bgarrido7.github.io/tese-app>. Here it is possible to access an exemplification of the application created. A detailed analysis of each screen is done below.

### 4.3.1 Main Menu Screen

On [Figure 4.12](#) a synopsis of the workflow can be seen, which will be described in detail below.

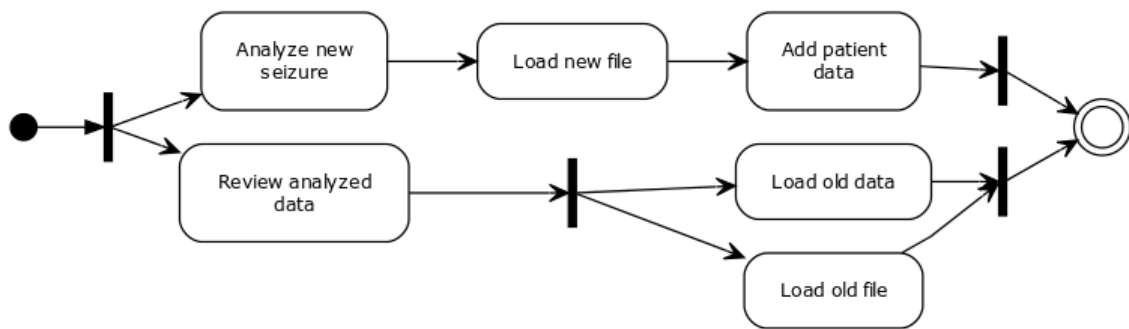


Figure 4.12: Activity diagram for the menu screen

Initially, the user is presented with the Main Menu page, available on [Figure A.1](#). Where it is possible to select a new seizure to analyze or an old one that already has data associated.

After selecting one, the pop-up from [Figure 4.13](#) is presented, where the user can select the seizure they wish to analyze.

Load File	
Name	Size
IM1198-2_sz01_kinect.zip	4 GB
IM1198-2_sz02_kinect.zip	5 GB
<b>IM1198-2_sz03_kinect.zip</b>	<b>10 GB</b>
IM1198-2_sz04_kinect.zip	6 GB
IM1198-2_sz05_kinect.zip	11 GB
IM1198-2_sz06_kinect.zip	3 GB
IM1198-2_sz07_kinect.zip	7 GB
IM1198-2_sz08_kinect.zip	5 GB

Close Ok

Figure 4.13: Load new seizure

The pop-up from the old seizure selection has the same appearance, only having the addition to select the data associated with the file chosen.

In case the user has selected a new analysis, the next pop-up presented on [Figure 4.14](#) will request them to fill the information regarding the selected seizure.

Figure 4.14: Patient data pop-up

Finally, once the “save” button is clicked, the user will automatically be redirected to the Tracking & Labeling page.

### 4.3.2 Tracking and Labeling Screen

This is the page where the user would spend most of their time, as it is the one where more interactions happen. Therefore, in order to have a better perception of the system functionalities, an overall view of the workflow of this complex page can be seen on [Figure 4.15](#).

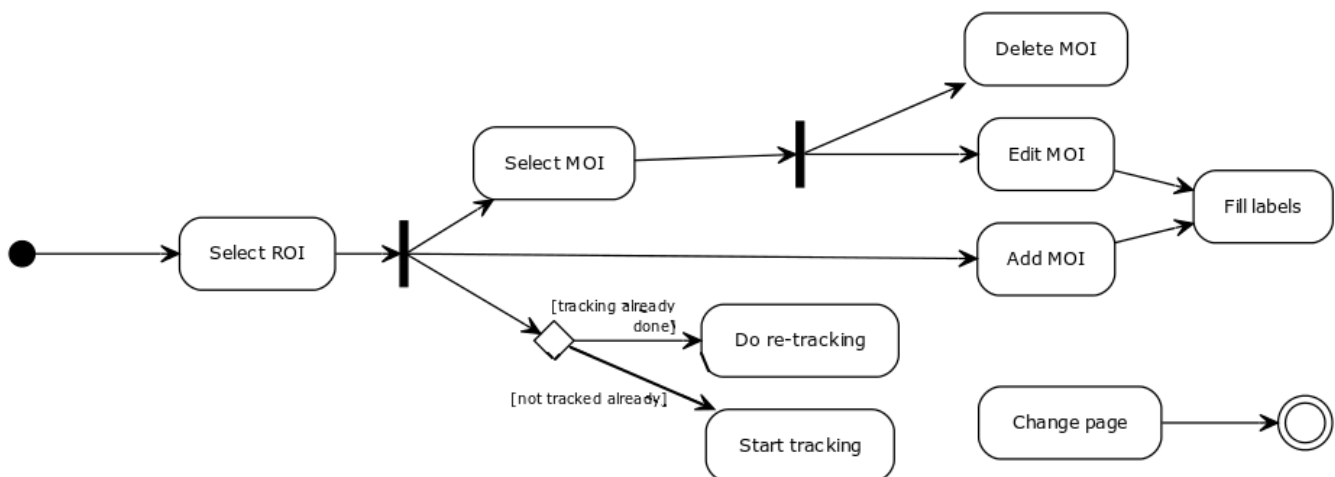


Figure 4.15: Activity diagram for the tracking &amp; labeling screen

First, when redirected to this screen, the user will be presented with an interface like the one from [Figure A.2](#). As we can notice, the main focus of this screen is the video part. In that regard, it is possible to check the raw image from the seizure and perform basic controls of the video (play, pause, stop, fast forward, wind back and skip by frame), thus fulfilling [Use Case 10](#).

A common component in every screen is the top bar, presented in [Figure 4.16](#). With this, it is possible to navigate to other pages namely the “About” page, where a short description of KiSA is presented, the “Account” page, that will be mentioned as future work, and other screens mentioned in this section.



Figure 4.16: Top bar

The way to navigate towards another page from this section is by using the “View” drop-down section in the top bar, or the arrows that appear in the middle, as displayed in [Figure 4.17](#).

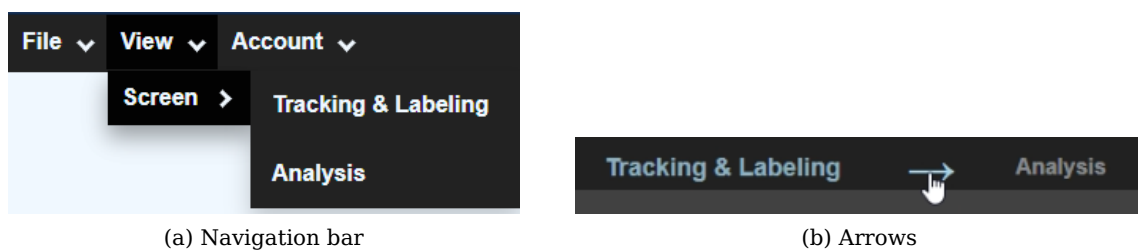


Figure 4.17: Change screen options

In addition, this screen also features the patient information that was inserted previously, displayed in [Figure 4.18](#).

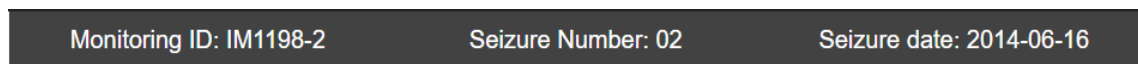


Figure 4.18: Patient data display

It is possible to edit this data just by using the “File” section in the top bar, as shown in [Figure 4.19](#). This will result in a pop-up appearing, very similar to the one in the Main Menu, but with the information already filled - refer to [Figure 4.20](#). If an alteration occurs the details from the patient data displayed at the top will also update, this way fulfilling [Use Case 2](#).

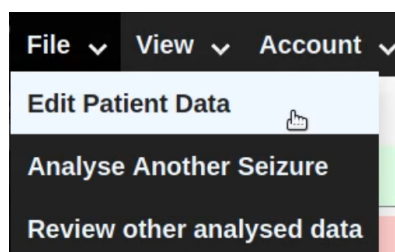


Figure 4.19: Select edit patient data

Also visible in this screen’s first appearance its the body parts relative to each ROI. Notice that when this section has a background with a red tone, it means that yet no tracking has been made for this ROI. On the contrary, if the color is similar to

Monitoring ID: IM1198-2      Seizure Number: 02

Monitoring ID \*  
IM1198-2

Seizure number \*  
02

Seizure date  
2014 - 06 - 16

Clinical Start  
10 : 02 : 18 p.m.

Clinical End  
10 : 02 : 38 p.m.

SAVE    CANCEL

Figure 4.20: Patient data filled

green it implies that the algorithm already monitored this body part, nevertheless, it is always possible to perform retracking. Notice that this implementation solves [Use Case 3](#).

In order to execute successful tracking and achieve [Use Case 5](#), the user must select a body part followed by the button to activate this process. This will trigger the appearance of the instructions presented on [Figure 4.21a](#) that will direct the user into marking the object desired and its vicinity, as can be seen on [Figure 4.21b](#).



Figure 4.21: Tracking process

Next, the user may choose to “Approve [the] Current ROI”. This process may be requested more than once since the algorithm is semi-automatic, it might lose the object location. Be that as it may, this process can take some time, in order to reassure the user that everything is working properly and nothing crashed, a load bar was developed as we can see from [Figure 4.22](#).

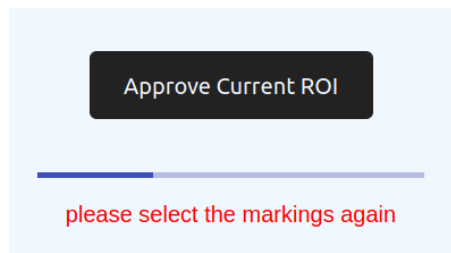


Figure 4.22: Tracking load bar

In the end, by the chance that the selected ROI had not been tracked yet, the background color relative to its section will turn green. Now, the video to be displayed is the same, with the change of a trace marking in the specific body part's movement, developed by the algorithm. As to be expected, every ROI that has this color will also show the trace of its movements as it is shown on [Figure 4.23](#) and [Figure 4.24](#), otherwise, just the normal video is presented, like in [Figure 4.25](#).



Figure 4.23: Right-Leg tracked video

As mentioned before, the software is perfectly capable of performing a retracking, if the user feels like it has not been done properly. In this case, the green section will remain the same color.

It is important to mention that the work described here is only executed with dummy data, due to the fact that the algorithm has been suffering some improvements, therefore not incorporated yet with the interface. But this simulation captures the essence of the workflow, giving the user a perfect notion of how they should expect the software to react, as it is the main intention of this thesis.

In order to make an attempt of accomplishing [Use Case 4](#), [Use Case 6](#), [Use Case 7](#), [Use Case 8](#), [Use Case 9](#), [Use Case 11](#) and [Use Case 12](#), a description of the necessary steps for MOI creation, labeling and editing is presented below.

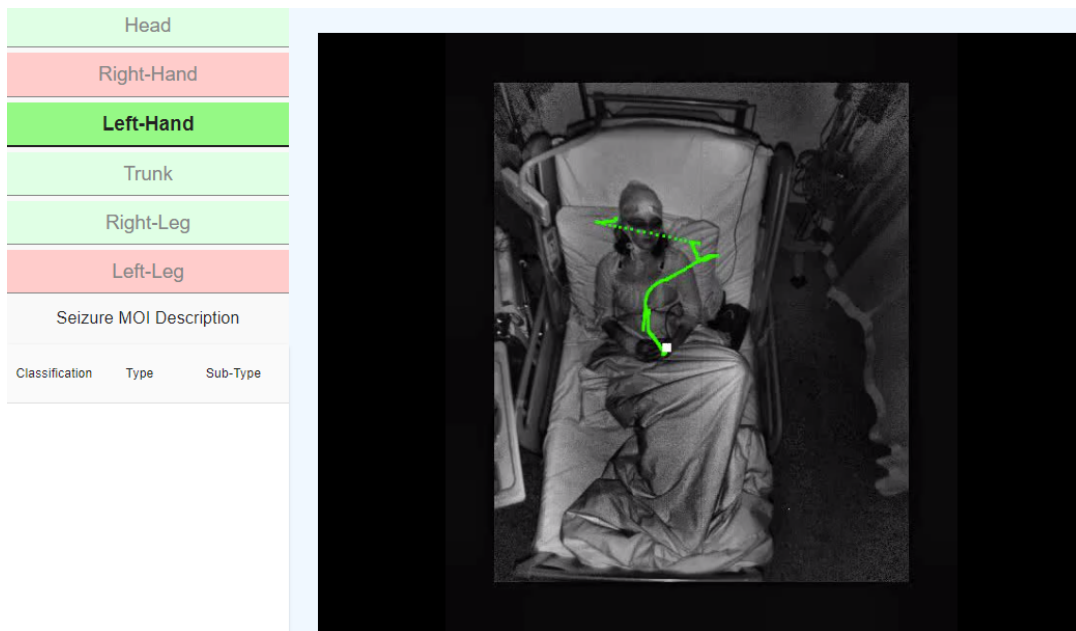


Figure 4.24: Left-Hand tracked video



Figure 4.25: Right-Hand not tracked video

Chosen the ROI, the user may select the option to “add MOI”. This action will trigger the apparition of the screen presented on [Figure A.3](#). Here it is possible to add the labels Classification, Type and Sub-type by choosing the desired value from each drop-down menu. Furthermore, the labels associated with clinical begin and clinical end can either be entered manually, by writing the correct numbers in the box input, or by sliding the timer thumb to the correct timestamp and select the radio box corresponding to the desired label.

It is important to notice that the Sub-type label depends on the previous one, therefore when the Type is chosen, the drop-down menu updates its content to match the correct options, as illustrated on [Figure 4.26](#).

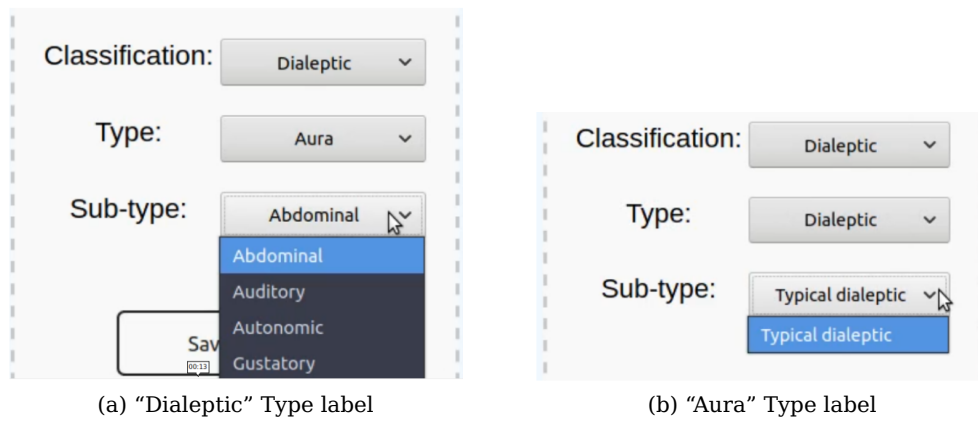


Figure 4.26: Sub-type label for different Types

After saving, the MOI is automatically added to the table. Here we can see that, when selected, the row becomes highlighted and the time slider emphasizes the interval of the specific MOI, from its clinical begin to the clinical end tags that were defined previously, as displayed on [Figure 4.27](#).



Figure 4.27: Time highlight when MOI selected

Additionally, we can try and modify each label. For this purpose, just by selecting a row from the table will trigger the screen from [Figure A.4](#), again the selection will be highlighted. From this point on, it is possible to completely eliminate the MOI, or edit one or more labels. Notice that the edit section is not available unless the user selects the "Modify" button. After performing the desired alterations, the "Save" option will make an update to the table, as well as to the highlight in the time slider in the event of updates regarding time labels occur.

Furthermore, the time highlight also serves the purpose of playing the video only in the period corresponding to the MOI occurrence, when choosing the "Play MOI" button. This feature was added in order for the doctors to have a better perception of the labels they have chosen before.

As it is understandable, the important matter to take from this section is the labeling of the MOI, since this data will be exported to the API, alongside with other indications regarding the state of the tracking and labeling from that session.

Moreover, it is significant to empathize that the MOI section is independent of the tracking, meaning that any body part can have MOI added, regardless of the ROI being tracked or not, as it possible to see in [Figure 4.28](#) and [Figure 4.29](#)

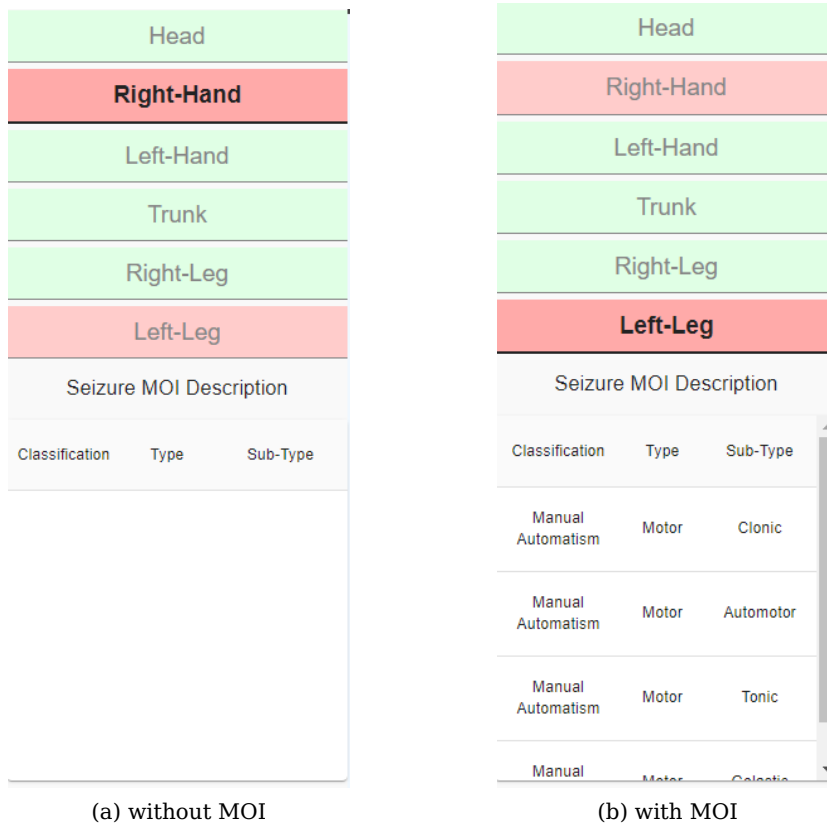


Figure 4.28: ROI with no tracking



Figure 4.29: ROI with tracking



### 4.3.3 Analysis Screen

The analysis page does not have many workflows, most of the components remain static regardless of the user interaction.

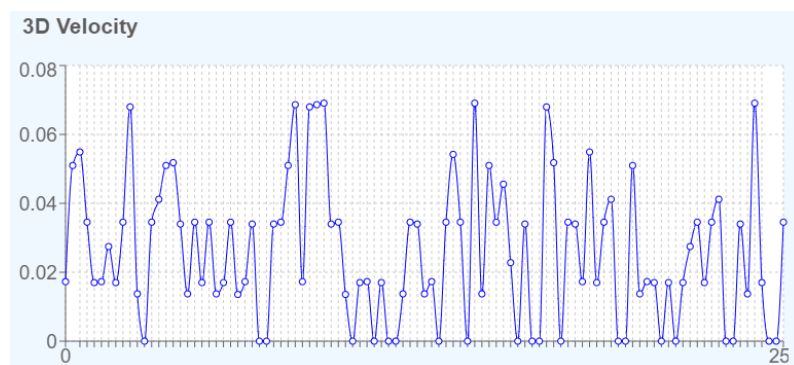
Again, it is possible to view the seizure video but on a smaller scale here, being that this page only has the purpose of displaying the gathered data.

As evidenced in [Figure A.5](#), this screen has three tables summing up the data acquired from the previous page. In the first it is possible to select multiple rows (as well as deselect them), corresponding to each ROI and its details, in order to filter the information displayed in the next table.

The second shows the supposed results computed from the algorithm, as well as a type of metrics available for analysis - velocity, jerk and acceleration -, these can be changed by selecting the drop-down located right above the table. The graph in the bottom left of the page will update according to the data chosen here, as we can see from [Figure 4.30](#) and [Figure 4.31](#). Since it is possible to have multiple rows from the previous table, it is feasible to observe some cross analytics.

MOI Quantification					
	Max	Min	Med	Mean	STD
Right-Hand-Velocity	0.3575	0	0.0362	0.0472	0.0618
Right-Hand-Velocity	0.3575	0	0.0362	0.0472	0.0618
Right-Hand-Velocity	0.3575	0	0.0362	0.0472	0.0618

(a) Table



(b) Graph

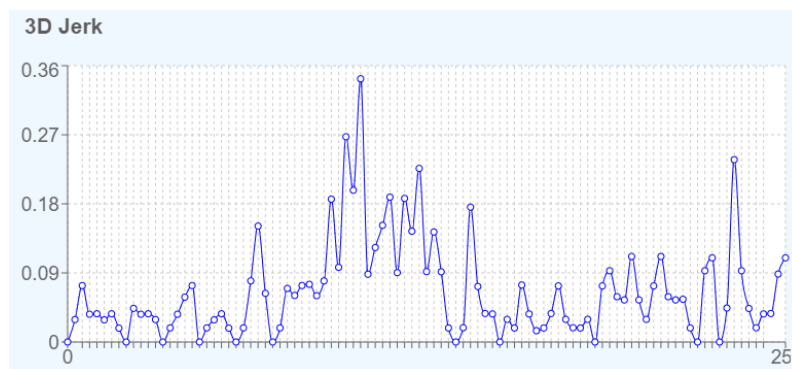
Figure 4.30: Analysis with velocity metric

Finally, the last table is an assemblage of details from the MOI selected in the previous board.

Once again, the information gathered here is only dummy data, since the algorithm is not incorporated and thus not having the correct output from the tracking and labeling details.

MOI Quantification					
	Max	Min	Med	Mean	STD
Right-Hand-Jerk	0.3575	0	0.0362	0.0472	0.0618
Right-Hand-Jerk	0.3575	0	0.0362	0.0472	0.0618
Right-Hand-Jerk	0.3575	0	0.0362	0.0472	0.0618

(a) Table



(b) Graph

Figure 4.31: Analysis with jerk metric

Furthermore, it is possible to navigate to the previous page, by clicking the arrow at the top of the screen, as well as in the “View” drop-down menu, likewise [Figure 4.17](#).

## 4.4 Usability Testing

Given that the most important requirement of this study is user-friendliness, it had to be found a way of verifying it, as well as the proper functioning of the interface.

Considering this, a survey was conducted with a random group of users, as well as doctors specialized in neurology, in order to evaluate the product by testing it with representative users. This approach is called Usability Testing [54].

For this survey, the interface was presented alongside with a script to follow. After every instruction completed, the user had to answer ten questions with five response options, on a scale from Strongly Disagree to Strongly Agree, as it can be seen on [Table 4.1](#). This questionnaire is inspired by the System Usability Scale (SUS), a tool for measuring usability originally created in 1986 [55, 56].

Below are the results of this survey, where the vertical and horizontal axis corresponds to the question number from the SUS and the total users that gave that answer, respectively.

Table 4.1: System Usability Scale Template

	Strongly Disagree		Strongly Agree	
1. I think that I would like to use this system frequently.				
2. I found the system unnecessarily complex				
3. I thought the system was easy to use.				
4. I think that I would need the support of a technical person to be able to use this system.				
5. I found the various functions in this system were well integrated.				
6. I thought there was too much inconsistency in this system.				
7. I would imagine that most people would learn to use this system very quickly.				
8. I found the system very cumbersome to use.				
9. I felt very confident using the system.				
10. I needed to learn a lot of things before I could get going with this system.				

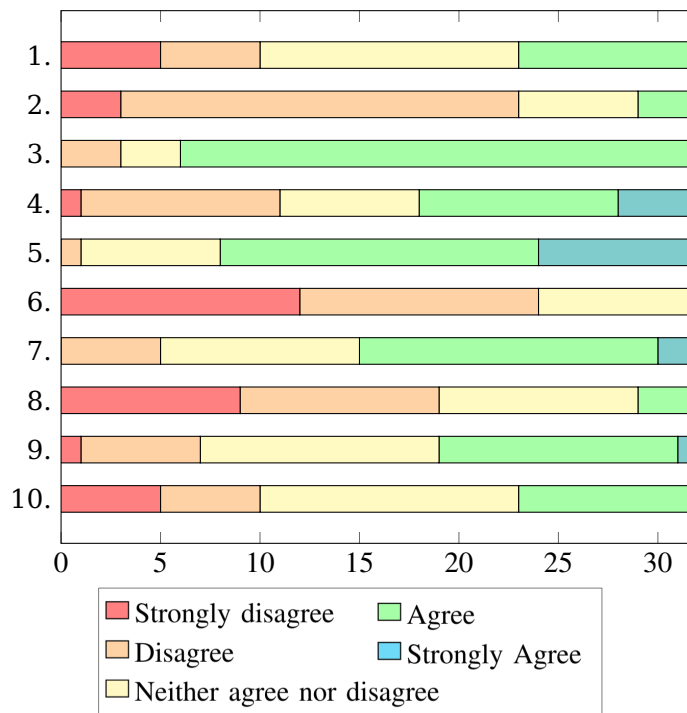


Figure 4.32: Survey results from regular users

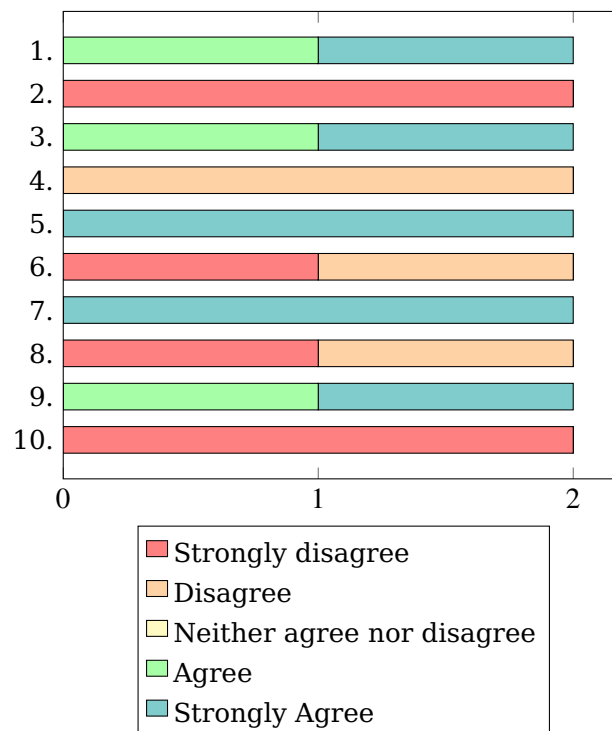


Figure 4.33: Survey results from medical users

The first survey results shown in [Figure 4.32](#) contain a sample of 32 average users, ages from 18 to 45 y/o, that claim they are either comfortable or very comfortable with technology and IT knowledge.

The survey was also sent to 5 doctors in Munich, Germany and 3 from Porto, Portugal. And the second results presented on [Figure 4.33](#) hold data acquired from 2 medical professionals that work on a specialty related to neurological diseases, ages from 26 to 45 y/o, that claim they are somewhat comfortable with technology and IT knowledge.

#### 4.4.1 Conclusions

The vast majority of users stated that the system was easy to use, had well-integrated features and was very consistent. The questions regarding future use are comprehensibly negative or neutral, due to this interface not being developed for average use, but rather design for a medical purpose.

## Chapter 5

# API Development and Testing

### 5.1 Implementation

As previously mentioned, it is necessary to develop a method for storing the information acquired from the user interaction. As clarified in [Section 3.1](#), since the application will be running on a web browser, it is necessary to make use of some HTTP request methods, in order to allow the communication between the front-end and API.

It is possible to understand this placement in the KiSA architecture from [Figure 5.1](#). The application retrieves data from the server and then the information gathered from the user's interaction is sent to the API, for this it is necessary to make use of the POST and PUT method, in order to store and update the data, respectively.

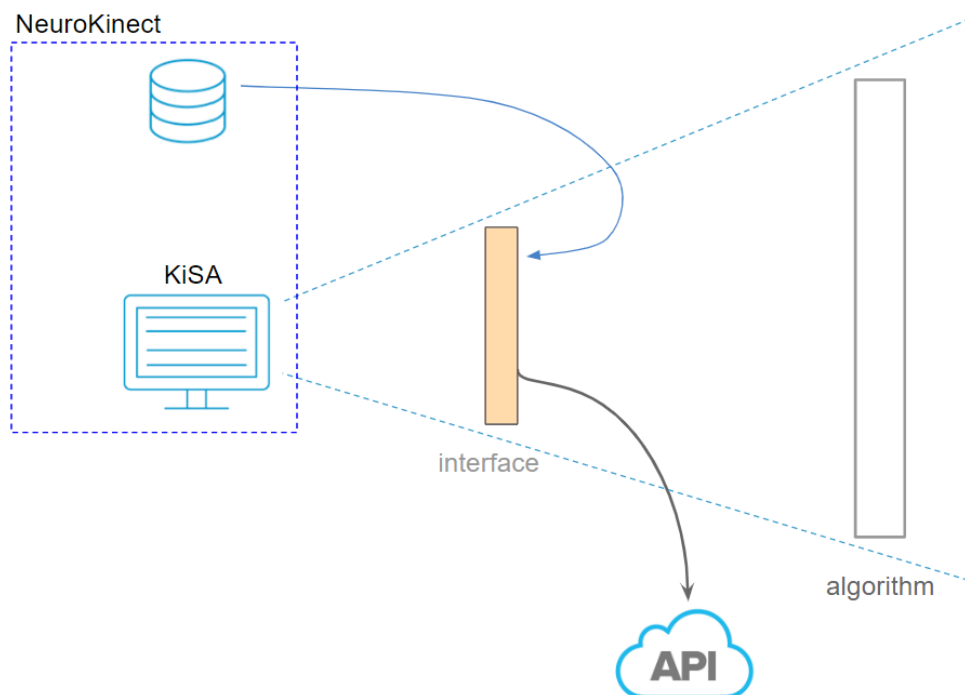


Figure 5.1: API placement in the KiSA architecture

Beyond the two HTTP methods discussed, the API can also handle a GET request. This was important to develop for the purpose of testing the system as an individual project and not a complementary feature of KiSA, in order to examine the correct behavior of the API.

The requirements to keep in mind for this part of the project are efficient communication between back-end and front-end, the flexibility of adding new features and fast response time. Having that in mind, Flask (a Python micro-framework) was chosen to develop the API, for the reasons already mentioned on [Subsection 3.3.1](#).

Once again, the API development was stored in the GitLab repository corresponding to the KiSA project. Nevertheless, the code for the API is also available on [Appendix B](#).

### 5.1.1 Data Sent

Once the user chooses a seizure to analyze (this data is acquired from the server), they need to fill in the sections corresponding to the patient's ID, Seizure Number, Data, Clinical Start and Clinical End. Alongside with these data, it is also sent information indicating if the chosen seizure was Tracked, Labeled, Needs Approval and Needs Review. These last two parameters are details from another work being developed in parallel with this thesis.

The way that KiSA calls the API is the following: when trying to exit the page, the user will be presented with a pop-up message confirming if they really wish to leave. When appearing, the application can detect if the data was already forward or if any alteration was made to it. It then proceeds to send a POST request, accompanied by a JSON array with the necessary information, if nothing was transmitted yet, or a PUT method with the new information updated, if the data was already but alterations were made.

After receiving a message from the client side, the API detects if a POST or PUT request was received, and proceeds to save the new data or update the last insertion, respectively. All of this information is stored in a JSON file, as mentioned in [Subsection 3.3.2](#), this format is the most appropriate to use due to space economizing and rapid reading. Additionally, the API also handles a GET request by sending the entire JSON array it has in store, this is relevant for future tests in [Section 5.3](#).

Granted that JSON is not a very secure file format, due to being a human-readable text, nevertheless, since authentication was not incorporated in this application, the security measures are not yet implemented, therefore it was not a concern in this stage of development. Furthermore, since the API has little communication with KiSA, in this point of progress, the tests made do not concern safeguards, since there are still other parameters to be incorporated in the data exchange.

### 5.1.2 Documentation

Alongside with the code development, it was created documentation to better understand and use the API. The [Table 5.1](#) shows the expected response for each method, either if the message is sent with success or if any error occurs.

Table 5.1: API documentation

Method	Success		Error	
	Code	Content	Code	Content
POST	201	"data inserted with success"	406	"incorrect JSON format"
PUT	202	"data updated with success"	406	"incorrect JSON format"
GET	200	JSON array	404	"file not found"

In addition, a requirements file was also developed, with the purpose of letting the user know which commands are necessary to run before using the API. The content of this file is detailed on [Listing 5.1](#).

```

1 click==7.1.2
2 Flask==1.1.2
3 Flask-API==2.0
4 Flask-Cors==3.0.8
5 itsdangerous==1.1.0
6 Jinja2==2.11.2
7 MarkupSafe==1.1.1
8 six==1.15.0
9 Werkzeug==1.0.1

```

Listing 5.1: Requirements File

## 5.2 Results

In the beginning, when tried to send an HTTP request from KiSA to the API, an error occurred regarding Cross-Origin Resource Sharing (CORS), a mechanism that uses additional HTTP headers<sup>1</sup> for the browser to allow the application to run on a server that has a different origin than the one that it is in.

Due to this problem, it can become difficult sometime to review the API, not knowing if an error occurred due to the browser or the code itself. Therefore, in the initial phase of the development, it is more practical to use Postman, a software development tool that acts as a client and calls the API just by entering the correct URL and the necessary body, if needed. Furthermore, this software will be also used for creating the API validation, as it will be shown in [Section 5.3](#). That being said, using this tool there is no need to worry about problems such as CORS or others that may come attached.

With the interest of running the API, it is necessary to have not only Python installed but also a virtual environment (venv), in order to deploy the application. This because, according to the Flask documentation<sup>2</sup>, the machine can have multiple

<sup>1</sup><https://developer.mozilla.org/pt-PT/docs/Web/HTTP/Headers>

<sup>2</sup><https://flask.palletsprojects.com/en/1.1.x/installation/>

Python versions installed and each can be used for various processes, so running the developed application can break compatibility in another project. That said, a **venv** was created using Python, where the application should run. Moreover, an address also needed to be created in order to call the API, that being said the one chosen was *localhost:5000/apiTest*.

In short, the main functionalities of the API can be seen in the following images [Figure 5.2](#), [Figure 5.3](#) and [Figure 5.4](#), where respectively is shown the results of a POST, followed by a GET and finally a PUT request. The first and last methods are required to send a specific body, this is possible to see in the terminal screen after it reads “POST received this”.

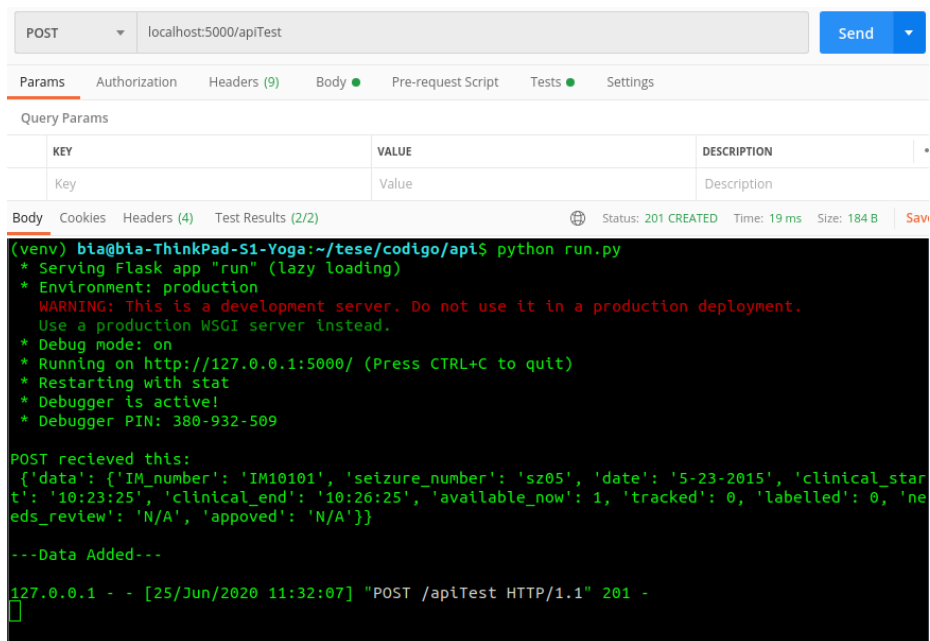


Figure 5.2: POST request sent to API

After the establishment of these results and the confirmation of the API’s correct behaviour, then the KiSA interface made a call to the API, using a POST and PUT method, in order to test the back-end and front-end communication. As described in [Subsection 5.1.1](#), a pop-up message will appear when the user tries to exit the application and here an HTTP method is sent. The results from this last test are the same as the ones from [Figure 5.4](#), minus the GET message.



GET localhost:5000/apiTest Send

Params Authorization Headers (9) Body Pre-request Script Tests Settings

Query Params

KEY	VALUE	DESCRIPTION
Key	Value	Description

Body Cookies Headers (4) Test Results (2/2) Status: 200 OK Time: 22 ms Size: 4.44 KB Save

```
(venv) bia@bia-ThinkPad-S1-Yoga:~/tese/codigo/api$ python run.py
* Serving Flask app "run" (lazy loading)
* Environment: production
  WARNING: This is a development server. Do not use it in a production deployment.
  Use a production WSGI server instead.
* Debug mode: on
* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
* Restarting with stat
* Debugger is active!
* Debugger PIN: 380-932-509

POST recieved this:
{'data': {'IM_number': 'IM10101', 'seizure_number': 'sz05', 'date': '5-23-2015', 'clinical_start': '10:23:25', 'clinical_end': '10:26:25', 'available_now': 1, 'tracked': 0, 'labelled': 0, 'need_s_review': 'N/A', 'approved': 'N/A'}}

---Data Added---

127.0.0.1 - - [25/Jun/2020 11:32:07] "POST /apiTest HTTP/1.1" 201 -

---Data Sent---

127.0.0.1 - - [25/Jun/2020 11:32:21] "GET /apiTest HTTP/1.1" 200 -
```

Figure 5.3: GET request sent to API

PUT localhost:5000/apiTest Send

Params Authorization Headers (9) Body Pre-request Script Tests Settings

Query Params

KEY	VALUE	DESCRIPTION
Key	Value	Description

Body Cookies Headers (4) Test Results (2/2) Status: 202 ACCEPTED Time: 11 ms Size: 184 B Save

```
(venv) bia@bia-ThinkPad-S1-Yoga:~/tese/codigo/api$ python run.py
* Serving Flask app "run" (lazy loading)
* Environment: production
  WARNING: This is a development server. Do not use it in a production deployment.
  Use a production WSGI server instead.
* Debug mode: on
* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
* Restarting with stat
* Debugger is active!
* Debugger PIN: 380-932-509

POST recieved this:
{'data': {'IM_number': 'IM10101', 'seizure_number': 'sz05', 'date': '5-23-2015', 'clinical_start': '10:23:25', 'clinical_end': '10:26:25', 'available_now': 1, 'tracked': 0, 'labelled': 0, 'need_s_review': 'N/A', 'approved': 'N/A'}}

---Data Added---

127.0.0.1 - - [25/Jun/2020 11:32:07] "POST /apiTest HTTP/1.1" 201 -

---Data Sent---

127.0.0.1 - - [25/Jun/2020 11:32:21] "GET /apiTest HTTP/1.1" 200 -

---Data Updated---

127.0.0.1 - - [25/Jun/2020 11:32:56] "PUT /apiTest HTTP/1.1" 202 -
```

Figure 5.4: PUT request sent to API

## 5.3 Functional Testing

For a more in depth analysis a more detailed test was done to for examine the correct behaviour of the system and status codes mentioned in [Table 5.1](#), once again using Postman in order to simulate the client side. That being said, the following scenario was created:

1. Sending a POST request with the following body:

```
1 {
2   data: {
3     IM_number: "IM9999",
4     seizure_number: "sz05",
5     date: "5-23-2015",
6     clinical_start: "10:23:25",
7     clinical_end: "10:26:25",
8     available_now: 1,
9     tracked: 0,
10    labelled: 0,
11    needs_review: "N/A",
12    approved: "N/A"
13  }
14 }
```

Listing 5.2: JSON Body for the first POST request

2. Next, a GET request is sent and the last insertion from the JSON array received, is compared with the "data" body that the previous message sent - [Listing 5.2](#) - to check if they are equal.
3. Afterwards, a PUT request is sent with the body:

```
1 {
2   data: {
3     IM_number: "IM10101",
4     seizure_number: "sz05",
5     date: "5-23-2015",
6     clinical_start: "10:23:25",
7     clinical_end: "10:26:25",
8     available_now: 1,
9     tracked: 0,
10    labelled: 0,
11    needs_review: "N/A",
12    approved: "N/A"
13  }
14 }
```

Listing 5.3: JSON Body for the second POST request

4. Another GET is sent, to confirm if the last element from the JSON file is equal to the previous body - [Listing 5.3](#). This way checking if the API updated the last inserted data.
5. Finally, in order to examine the behavior of a faulty message, a POST and PUT requests are sent, separately, along with a body having an incorrect name, instead of “data” the message sends a body with “wrong\_data”.

As expected, all the tests were successful, as we can see from [Figure 5.5](#). Furthermore, the code numbers from each request message also match the code indicated in the documentation [Table 5.1](#).

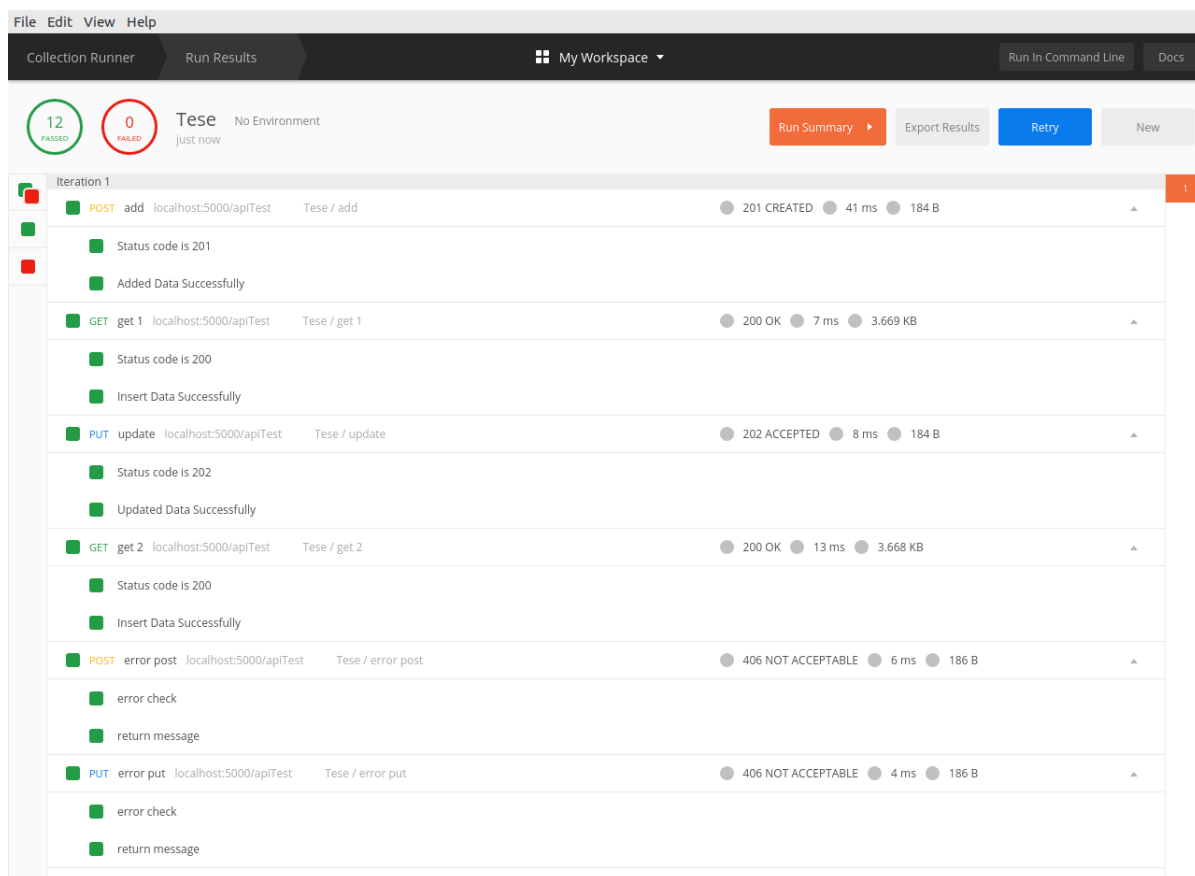


Figure 5.5: API test results

As we can see from the previous image, as well as the ones in the previous section, the response is fairly quick. But in order to have a better perception of the system’s performance, a study was conducted using the same methodology, but with five different runs, with the intent of calculating an average time for each method. The time unit for each test is shown in milliseconds (ms).

From [Table 5.2](#) we can conclude that efficiency and speed requirements were well accomplished, having a time delay of 10.4ms for the POST method, 6.8ms for the GET method and 8.8ms for the PUT method.

Table 5.2: Response time for each HTTP method developed

Run #	Test method		
	POST response time	GET response time	PUT response time
1	10	6	6
2	15	7	8
3	9	6	8
4	10	7	10
5	8	8	12
Average	10.4	6.8	8.8

## Chapter 6

# Conclusions and Future Work

This chapter is a reflection on the entire work done in this thesis. An overview of the requirements established in the beginning and their respective accomplishment. Followed by the possible future work that could improve even more the system and make KiSA the desirable tool for Epilepsy assessment.

### 6.1 Overview

Since the programming language changes, the software has not only a performance boost but also the freedom to develop a better GUI. As a result, the new interface has a much faster response to the user interaction and a more intuitive and appealing visual aspect.

Reviewing the requirements established on [Section 2.5](#), this study needed to develop a system that had:

- Fast response time
- Efficient communication between back-end and front-end
- User-friendliness
- Usability
- Flexibility of adding new features

The biggest point to check was the user-friendliness. This point was accomplished since the specialist from HSJ that gave his contribution and opinion on this work was very pleased with the final result. Furthermore, the survey results from [Section 4.4](#) also gave good feedback about the user experience and satisfying intuitive system, not only with average users but also with medical professionals, which will be the target audience for this application.

Moreover, the issues regarding system performance were also improved, given that the interface has a fast response time to the user interactions. In addition, the latency from the API is also very small, as concluded on [Section 5.2](#), giving a contribution to not only response time but also effective communication between front-end and back-end.

Finally, considering that the code was developed with React, it is very easy to implement new features. Being with reusing components already implemented, or importing other elements from resources available online.

## 6.2 Future Work

This thesis is a part of a bigger project that is KiSA. Since this study mainly focuses on interface development, there are still many components to be integrated and more features can always be added.

The main aspect that was not included in this study, was the algorithm incorporation. The tracking part of the system developed here has no realistic output, just a simulation of how the interface would react if the algorithm was implemented. Furthermore, the metrics that appear on the analysis screen are also static, they do not reflect realistic data from the seizure analysis since the information that is shown in the tables of this page is all dummy data.

Moreover, the initial step of KiSA, choosing a seizure to analyze, also needs to be associated with real data deriving from a database or storage unit, since now it is only shown as dummy data. This real data refers not only to the seizure video file but also labels associated with each episode. Some of these labels can be incorporated in the API developed in this study, as there is already a slot reserved for them - the "needs review" and "needs approval" labels.

Finally, when the proper data from the analysis screen is incorporated, it would be pleasant to export the metric computed, in order to continue a future assessment outside the KiSA system. A suggestion for a file format for this exported data is CSV since most doctors are already accustomed to viewing data in this format.

Regarding additional features, there are some that are being developed in parallel with this study, as such, it would not be viable to incorporate yet since they are still in the experimental phase. With this in mind, a suggestion of some User Stories to be implemented in future iterations of KiSA are:

- **Use Case 1:** Change the primary source (simple depth, infrared, 3D point cloud, etc.)
- **Use Case 2:** See a percentage of parts manually tracked
- **Use Case 3:** Export the data computed
- **Use Case 4:** Visualize a 3D version of the video tracked

Furthermore, it would be desired in the future to associate each interaction with a specific user, in order to better criticize the labeling and reviewing made in each session. Hence, a login and register system would be a necessary feature to have in the long run. With this in mind, this thesis already developed a front-end page that can be used as a draft for future implementations of user sessions.

In short, considering the goal of making KiSA an easier to use tool for doctors, many of the tasks mentioned in this section focus on making the system even more intuitive and overall user-friendly.

In a thesis that set out to solve a UI problem with new and enhance technologies, one could say that all of the proposed items were fulfilled with success, achieving the goals establishes for this work.







# Appendix A

## Interface Results

### A.1 Main Menu

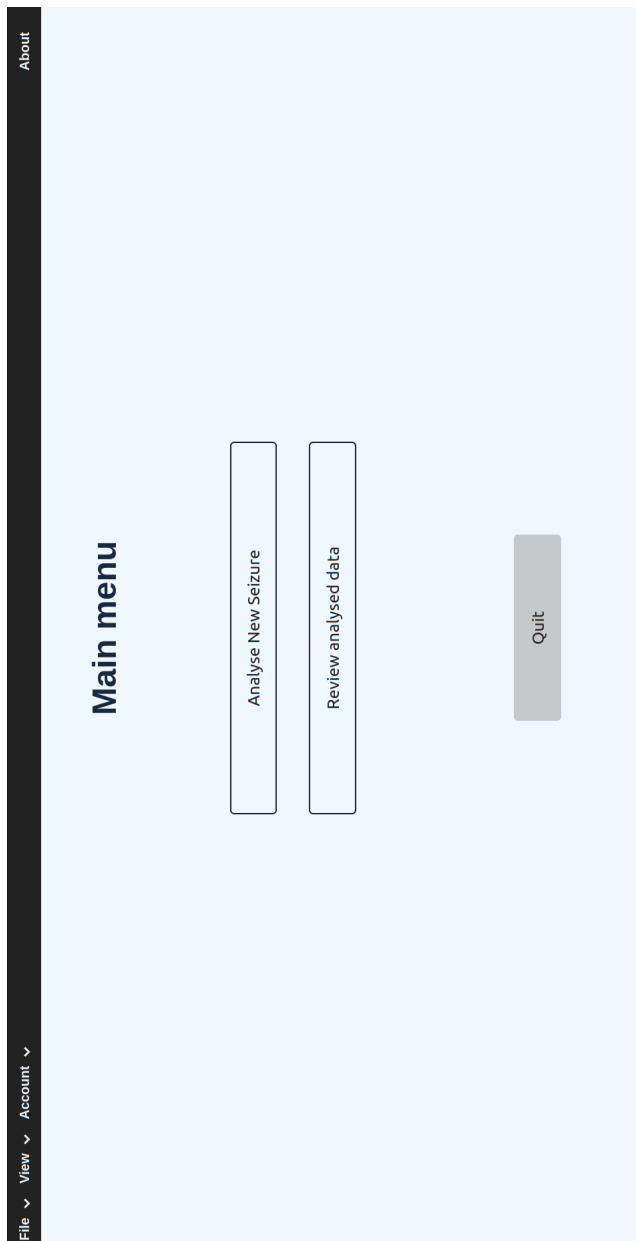


Figure A.1: Main Menu Screen

## A.2 Tracking and Labeling



Figure A.2: Tracking & Labeling Screen - Nothing Selected

File View Account

Tracking & Labeling

About

**ROI**

- Head
- Right-Hand
- Left-Hand
- Trunk
- Right-Leg
- Left-Leg**

Monitoring ID: IM1198-2


Seizure Number: 02

Seizure date: 2014-06-16

Seizure MOI Description

Classification	Type	Sub-Type
Manual Automatism	Motor	Clonic
Manual Automatism	Motor	Automotor
Manual Automatism	Motor	Tonic

Add MOI



⏪
⏴
⏵
⏩
⏸
⏹

Timestamp: 00:000

Clinical Begin

Clinical End

On Current Timestamp

Classification:

Type:

Sub-type:

Save MOI

Figure A.3: Tracking & Labeling Screen - Add Selected

File > View > Account >
Tracking & Labeling →
About

Monitoring ID: IM1198-2

Seizure date: 2014-06-16

Seizure Number: 02

**ROI**

Head

Right-Hand

Left-Hand

Trunk

Right-Leg

**Left-Leg**

Seizure MOI Description	
Classification	Type Sub-Type
Manual Automatism	Motor Clonic
Manual Automatism	Motor Automotor
Manual Automatism	Motor Tonic

Delete MOI
Add MOI

Timestamp: 00:000

**Modify MOI**

Clinical Begin: 05:440  On Current Timestamp

Clinical End: 08:450

Classification: Manual Automatism

Type: Motor

Sub-Type: Automotor

**Save MOI**

**Play MOI**

Figure A.4: Tracking & Labeling Screen - Edit Selected

### A.3 Analysis

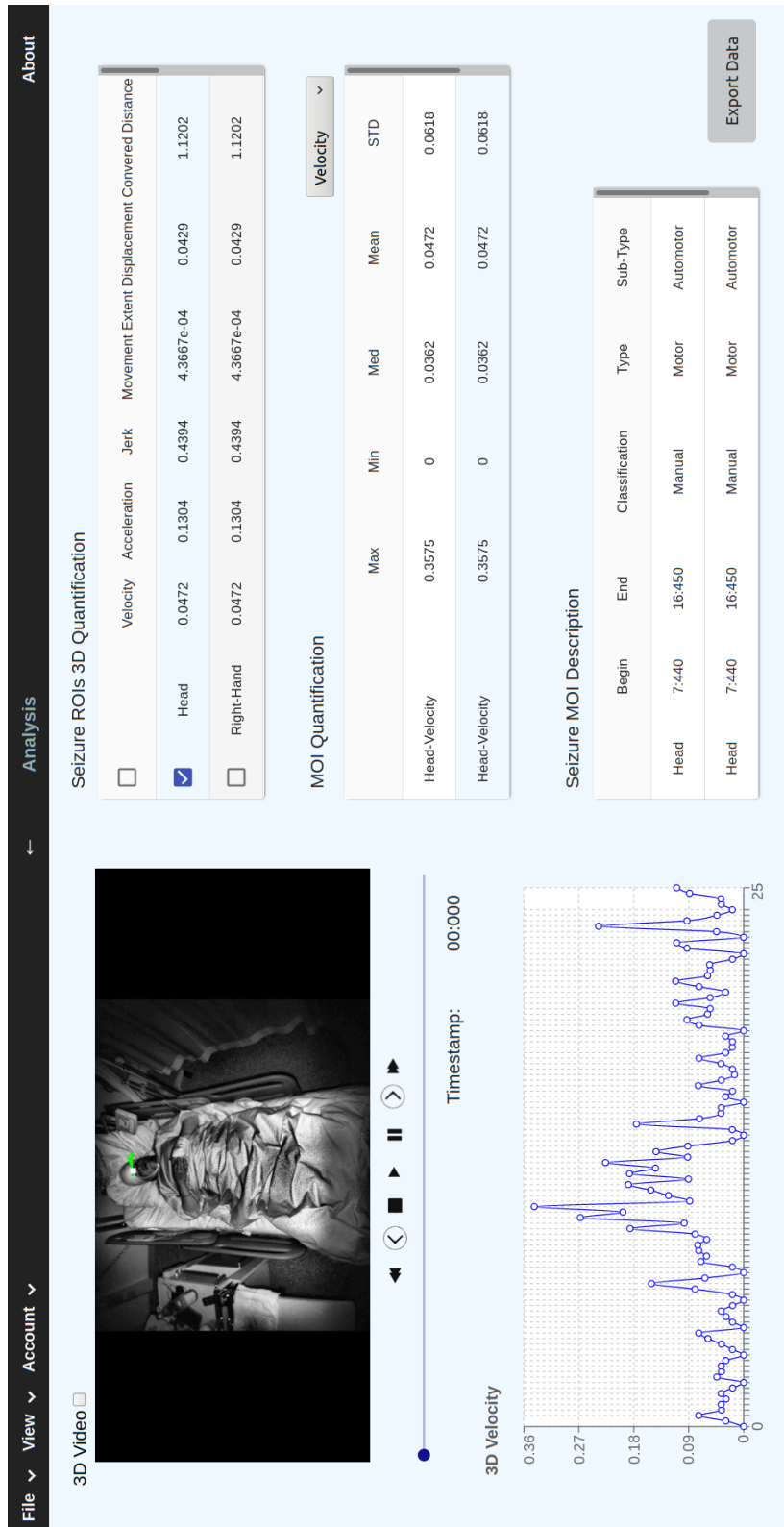


Figure A.5: Analysis Screen

## Appendix B

# API Code

```
1  from flask import Flask, render_template, jsonify, request, json
2  from flask_cors import CORS, cross_origin
3  from flask_api import FlaskAPI, status, exceptions
4  import os
5
6  app = Flask(__name__)
7  CORS(app)
8
9  @app.route('/apiTest', methods=['GET', 'POST', 'PUT'])
10 @cross_origin(supports_credentials=True)
11 def update_data():
12
13     if request.method == 'POST':    # add new data
14         print("\nPOST recieved this:\n", request.json)
15
16         if "data" not in request.json:
17             return "incorrect json format", status.HTTP_406_NOT_ACCEPTABLE
18         else:
19             with open('./data.json') as json_file:
20                 data = json.load(json_file)
21                 data.append(request.json['data'])
22
23
24             with open('./data.json','w') as f:
25                 json.dump(data, f, indent=4)
26                 print("\n---Data Added---\n"),
27                 return "data inserted with success", status.HTTP_201_CREATED
28
29     if request.method == 'PUT':    # update data
30
31         if "data" not in request.json:
32             return "incorrect json format", status.HTTP_406_NOT_ACCEPTABLE
33
34         else:
35
36             with open('./data.json') as json_file:
37                 data = json.load(json_file)
```

```
38         data[-1] = request.json['data']
39
40     with open('./data.json','w') as f:
41         json.dump(data, f, indent=4)
42     print("\n---Data Updated---\n")
43
44     return "data updated with success", status.HTTP_202_ACCEPTED
45
46     else:
47         if not os.path.exists('./data.json'):
48             print("\n---File Not Found--\n")
49             return "file not found", status.HTTP_404_NOT_FOUND
50
51         else:
52             with open('./data.json', 'r') as jsonfile:
53                 file_data = json.loads(jsonfile.read())
54
55             print("\n---Data Sent---\n")
56             return json.dumps(file_data), status.HTTP_200_OK
57
58
59
60 if __name__ == '__main__':
61     app.run(debug=True)
62
63 if __name__ == '__main__':
64     flask_cors.CORS(app, expose_headers='Authorization')
```

Listing B.1: Code developed for the API



# References

- [1] Paulo Maia, Elisabeth Hartl, Christian Vollmar, Soheyl Noachtar, and João Paulo Cunha. Epileptic seizure classification using the neuromov database. pages 1–4, 02 2019. doi:10.1109/ENBENG.2019.8692465.
- [2] F.F. Ferri. *Ferri's Clinical Advisor 2019 E-Book: 5 Books in 1*. Ferri's Medical Solutions. Elsevier Health Sciences, 2018. Available: <https://books.google.pt/books?id=-L5dDwAAQBAJ>.
- [3] Yaejin Moon, Jonghun Sung, Ruopeng an, Manuel Hernandez, and Jacob Sosnoff. Gait variability in people with neurological disorders: A systematic review and meta-analysis. *Human Movement Science*, 47:197–208, 06 2016. doi:10.1016/j.humov.2016.03.010.
- [4] Hugo Miguel Pereira Choupina. Neurokinect: Kinect-based system for motion analysis and quantification in neurological diseases. Master's thesis, Faculdade de Engenharia da Universidade do Porto, Portugal, 2014. Available: <https://hdl.handle.net/10216/85948>.
- [5] Joana Catarina Moreira Rodrigues. Gaitgate - towards a multi-scenario clinical gait characterization system for neurological diseases. Master's thesis, Faculdade de Engenharia da Universidade do Porto, Portugal, 2019. Available: <https://hdl.handle.net/10216/123784>.
- [6] Hugo Miguel Pereira Choupina, Ana Patrícia Rocha, José Maria Fernandes, Christian Vollmar, Soheyl Noachtar, and João Paulo Silva Cunha. NeuroKinect 3.0: Multi-bed 3Dvideo-EEG system for epilepsy clinical motion monitoring. *Studies in Health Technology and Informatics*, 247:46–50, 2018. doi:10.3233/978-1-61499-852-5-46.
- [7] Tamás Karácsony, Vitor Minhoto, and Joao Paulo Silva Cunha. An Epileptologist-Friendly Cloud-Based Remote 3Dvideo-EEG Processing Environment For Quantified Semiology Analysis. pages 1–2, 01 2020. Available at: <https://www.researchgate.net>.
- [8] João Paulo Cunha, Hugo Choupina, Ana Rocha, Jose Maria Fernandes, Felix Achilles, Anna Loesch, Christian Vollmar, Elisabeth Hartl, and Soheyl Noachtar. Neurokinect: A novel low-cost 3dvideo-eeG system for epileptic seizure motion quantification. *PLoS one*, 11, 01 2016. doi:10.1371/journal.pone.0145669.
- [9] Jason Williams. Difference between framework vs library vs ide vs api vs sdk vs toolkits? Available at: <https://stackoverflow.com>. [Last Updated Sep 21, 2017].

- [10] Natanael Silva Cardoso and Thamirys Martha da Silva Bispo. Um estudo comparativo entre os principais frameworks de desenvolvimento web em linguagem python. 2019. Available: [bdta.ufra.edu.br/jspui/handle/123456789/541](http://bdta.ufra.edu.br/jspui/handle/123456789/541).
- [11] MDN web docs. HTTP. Available: <https://developer.mozilla.org/en-US/docs/Web/HTTP>. [Last Updated May 18, 2020].
- [12] MDN web docs. HTTP request methods. Available: <https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods>. [Last Updated Feb 1, 2020].
- [13] Postman. What is postman, and how do i set-up the tool? Available: <https://kb.uwm.edu/uwmhd/page.php?id=95041>. [Last Updated Oct 16, 2019].
- [14] Margaret Rouse. What is native app? Available: <https://searchsoftwarequality.techtarget.com/definition/native-application-native-app>. [Last Updated March, 2018].
- [15] Python.org. Applications for python. Available: <https://www.python.org/about/apps/>.
- [16] D.B. Beniz and A.M. Espindola. Using Tkinter of Python to Create Graphical User Interface (GUI) for Scripts in LNLs. In *Proc. of International Workshop on Personal Computers and Particle Accelerator Controls (PCaPAC'16), Campinas, Brazil, October 25-28, 2016*, number 11 in International Workshop on Personal Computers and Particle Accelerator Controls, pages 56–58, Geneva, Switzerland, Sep. 2017. JACoW. doi:10.18429/JACoW-PCaPAC2016-WEPOPRPO25.
- [17] Jinwei Lin and Aimin Zhou. Pydraw: a gui drawing generator based on tkinter and its design concept. *ArXiv*, 2018. arXiv:1808.09094.
- [18] CtechF. Python tutorial: Python gui programming (tkinter). Available: <https://web.archive.org/web/20191210153336/https://ctechf.com/python-gui-programming>. [Last Updated June 9, 2019].
- [19] Ambika Choudhury. 8 python gui frameworks for developers. Available: <https://analyticsindiamag.com/8-python-gui-frameworks-for-developers/>. [Last Updated Sept 18, 2019].
- [20] Python Wiki. PyQt. Available: <https://wiki.python.org/moin/PyQt>. [Last Updated Jun 29 ,2020].
- [21] Michael Herrmann. PyQt5 tutorial 2020: Create a gui with python and Qt. Available: <https://build-system.fman.io/pyqt5-tutorial>.
- [22] 2020 The Qt Company Ltd. Getting to know Qt designer. Available: <https://doc.qt.io/qt-5/designer-to-know.html>.
- [23] The wxPython Team. Overview of wxPython. Available: <https://wxpython.org/pages/overview/index.html>.
- [24] The wxPython Team. Thumbnail gallery — wxPython Phoenix 4.1.0a1 documentation. Available: <https://wxpython.org/Phoenix/docs/html/gallery.html>.

- [25] Ondřej CHRASTINA. Cross-platform development of smartphone application with the kivy framework [online]. Master's thesis, Masaryk University, Faculty of Informatics, Brno, 2016 [cit. 2020-02-07]. [theses.cz/id/t7010i/](https://theses.cz/id/t7010i/).
- [26] Aman Bhoyarkar, Anuja Solanki, and Ashutosh Balbudhe. Application development using kivy framework. *IJARCCCE*, 8:53–58, 02 2019. doi:10.17148/IJARCCCE.2019.8209.
- [27] Kenneth Reitz. Gui applications — the hitchhiker's guide to python, 2011-2020. Available: <https://docs.python-guide.org/scenarios/gui/>.
- [28] Irina Kravchenko. React vs angularjs comparison: Which is better in 2019? Available: <https://diceus.com/react-vs-angularjs/>. [Last Updated Sept 19, 2019].
- [29] Oleg Romanyuk. Angular vs react: Which one to choose for your app. Available: <https://www.freecodecamp.org/news/angular-vs-react-what-to-choose-for-your-app-2/>. [Last Updated Oct 8, 2019].
- [30] React native · a framework for building native apps using react. Available: <https://reactnative.dev/>.
- [31] Tonya Smyrnova. Angular vs react vs vue battle with pros and cons for all of them. Available: <https://syndicode.com/2019/04/15/angular-vs-react-vs-vue-battle-with-pros-and-cons-for-all-of-them/>. [Last Updated April 15, 2019].
- [32] Maximilian Schwarzmüller. Angular vs react vs vue - my thoughts. Available: <https://academind.com/learn/javascript/angular-vs-react-vs-vue-my-thoughts/>, 2017. [Last Updated May 15, 2017].
- [33] Comparison with other frameworks — vue.js. Available: <https://vuejs.org/v2/guide/comparison.html>.
- [34] Django Software Foundation. The web framework for perfectionists with deadlines. Available: <https://www.djangoproject.com/>.
- [35] Suryadiputra Liawatimena, Harco Leslie Hendric Spits Warnars, Agung Triset-yarso, Edi Abdurahman, Benfano Soewito, Antoni Wibowo, Ford Gaol, and Bah-tiar Abbas. Django web framework software metrics measurement using radon and pylint. pages 218–222, 09 2018. doi:10.1109/INAPR.2018.8627009.
- [36] A. Yim, C. Chung, and A. Yu. *Matplotlib for Python Developers: Effective techniques for data visualization with Python, 2nd Edition*. Packt Publishing, 2018. Available: <https://books.google.pt/books?id=G99YDwAAQBAJ>.
- [37] GeeksforGeeks. Mvc design pattern. Available: <https://www.geeksforgeeks.org/mvc-design-pattern/>.
- [38] Jan PATER. Moderní webové aplikační frameworky [online]. Master's thesis, Masaryk University, Faculty of Informatics, Brno, 2016 [cit. 2020-02-06]. Available: <https://is.muni.cz/th/uz7ba/>.

- [39] Ryan Brown. Django vs flask vs pyramid: Choosing a python web framework. Available: <https://www.airpair.com/python/posts/django-flask-pyramid>.
- [40] Patrick Vogel, Thijs Klooster, Vasilios Andrikopoulos, and Mircea Lungu. A low-effort analytics platform for visualizing evolving flask-based python web services. pages 109–113, 09 2017. doi:10.1109/VISSOFT.2017.13.
- [41] W3Schools. JSON vs XML. Available: [https://www.w3schools.com/JS/js\\_json\\_xml.asp](https://www.w3schools.com/JS/js_json_xml.asp).
- [42] Nurzhan Nurseitov, Michael Paulson, Randall Reynolds, and Clemente Izurieta. Comparison of json and xml data interchange formats: A case study. pages 157–162, 01 2009. Available: <https://www.semanticscholar.org>.
- [43] Bianka Pluszczewska. 5 best javascript frameworks for desktop apps. Available: <https://brainhub.eu/blog/javascript-frameworks-for-desktop-apps/>.
- [44] Ashutosh KS. Frameworks & tools to develop cross-platform desktop apps – best of. Available: <https://www.hongkiat.com/blog/frameworks-tools-build-cross-platform-desktop-apps/>. [Last Updated Nov 23, 2017].
- [45] About - proton native - react native for the desktop, cross compatible. Available: <https://proton-native.js.org/#/about>.
- [46] Epilepsy Foundation Research. 2020 epilepsy pipeline conference. Available: <https://www.epilepsy.com/make-difference/research-and-new-therapies/engagement/2020-epilepsy-pipeline-conference>. [Last Updated May, 2020].
- [47] Jonathan J. Halford, Deng Shan Shiau, Ryan T. Kern, Conrad A. Stroman, Kevin M. Kelly, and J. Chris Sackellares. Seizure detection software used to complement the visual screening process for long-term EEG monitoring. *Neurodiagnostic Journal*, 50(2):133–147, 2010. doi:10.1080/1086508x.2010.11079764.
- [48] Maki.vc. Meet neuro event labs: Quantifying epileptic seizures with computer vision and AI. Available at: <https://medium.com/>. [Last Updated Jun 13, 2018].
- [49] Go North Medical Neuro. General 4. Available: <https://www.gonorthmedical.com/neuro-event-labs>.
- [50] Sinimuna. Nelli dashboard. Available: <https://sinimuna.com/project/nelli/>.
- [51] Epihunter. Brainlink EEG headset. Available: <https://www.epihunter.com/brainlink-headset>.
- [52] Epihunter. Accurate seizure overview, with video! Available: <https://www.epihunter.com/professionals>.
- [53] Epihunter. epihunter companion. Available: <https://www.epihunter.com/en/companion-app>.
- [54] Usability.gov. Usability testing. Available: <https://www.usability.gov/how-to-and-tools/methods/usability-testing.html>.

- [55] Georgia Gallavin. System usability scale (SUS): Improving products since 1986. Available: <https://digital.gov/2014/08/29/system-usability-scale-improving-products-since-1986/>. [Last Updated Aug 29, 2014].
- [56] Usability.gov. System usability scale (SUS). Available: <https://www.usability.gov/how-to-and-tools/methods/system-usability-scale.html>.