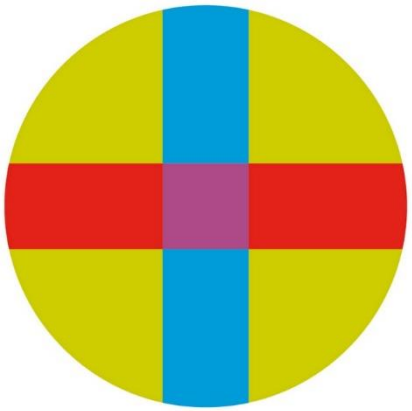
UNIVERSITY CEU - SAN PABLO

POLYTECHNIC SCHOOL

BIOMEDICAL ENGINEERING DEGREE



BACHELOR THESIS

MICROSERVICE ARCHITECTURE DESIGN FOR SUPPORTING PERSONALIZED REHABILITATION

Author: Sofía G. Grandía
Supervisors: Sergio Saugar García and Rodrigo García Carmona

February 2022

UNIVERSIDAD SAN PABLO-CEU
ESCUELA POLITÉCNICA SUPERIOR
División de Ingeniería

Datos del alumno

NOMBRE: Sofía González Grandía

Datos del Trabajo

TÍTULO DEL PROYECTO: Microservice Architecture design for supporting personalized rehabilitation

Tribunal calificador

PRESIDENTE:	FDO.:
SECRETARIO:	FDO.:
VOCAL:	FDO.:

Reunido este tribunal el _____/_____/_____, acuerda otorgar al Trabajo Fin de Grado presentado por Don_____la calificación de _____.

ACKNOWLEDGMENTS

Después de haber acabado el trabajo, resulta que esta va a ser la sección que más me va a costar escribir, ya que significa decirle adiós a la Universidad CEU San Pablo y todo lo que me ha dado.

Gracias a todas las personas que me han ayudado con este trabajo, y en general durante toda la carrera, empezando por mi madre. Gracias a su paciencia infinita (y a su insistencia que ha llegado a acabar con mi paciencia limitada) he conseguido llegar a este punto. A José Ramón, que tenía la certeza de que iba a llegar aquí antes de que lo supiese yo. A mi hermana Paula, gracias por todo.

A mis tutores, Sergio y Rodrigo, quienes han persistido y aguantado conmigo y con este TFG. Gracias por vuestra disponibilidad, conocimiento y perseverancia. A absolutamente todos y cada uno de los profesores que he tenido durante la carrera, gracias. Todos me habéis enseñado e influenciado de una manera que me va a marcar eternamente. Agradezco vuestra cercanía y formación. No os lo puedo agradecer más.

A mis compañeros de clase. No podría haber tenido más suerte. Gracias a todos por crear un ambiente tan cómodo y de compañerismo. Gracias a mis amigas, sin las cuales no estaría aquí escribiendo agradecimientos. Por estar siempre presentes y ser la motivación del día a día.

ABSTRACT

Cerebral Palsy is one of the most common motor disabilities, currently affecting 2-3 children out of 1000. This disorder affects posture and movement, and it has been proven that physical therapy can improve the balance and gait of those who suffer from this condition. Rehabilitation can be aided with the use of an exoskeleton. These devices not only help in correcting motor control, but they also allow for the measurement and recording of muscle electromyography values, as well as comparison between therapy sessions of the same patient.

The values obtained during a therapy session can be stored and used to personalise the rehabilitation process of a patient, making it more effective. The purpose of this project is to correctly and effectively store the ample amounts of data generated during a therapy session using a non-relational database, as well as creating an expert system that will help decide the therapeutic route of the patient using the information from the database and the data inserted by the therapist. These two pillars of the project will be able to communicate with each other using web service technologies, as well as other specific state of the art tools. The architecture is done using web services so that it can then be used in rehabilitation video games that are used with the exoskeleton. Then the configuration of the game, such as levels and difficulty, will be based on the historical information of the patient.

RESUMEN

La parálisis cerebral es una de las discapacidades motrices más frecuentes, y actualmente afecta a 2 - 3 niños de cada 1.000. Este trastorno afecta a la postura y el movimiento, y se ha demostrado que la fisioterapia puede mejorar el equilibrio y la marcha de quienes la padecen. La rehabilitación puede ser asistida con el uso de un exoesqueleto. Estos dispositivos no sólo ayudan a corregir el control motor, sino que también permiten medir y registrar los valores de electromiografía muscular, así como comparar entre sesiones de terapia del mismo paciente.

Los valores obtenidos durante una sesión de terapia pueden almacenarse y utilizarse para personalizar el proceso de rehabilitación de un paciente, mejorando su eficacia. El objetivo de este proyecto es almacenar de forma eficiente la gran cantidad de datos generados durante una sesión de terapia utilizando una base de datos no relacional, así como crear un sistema experto. Dicho sistema podría ayudar al profesional a decidir la ruta terapéutica más adecuada del paciente combinando la información extraída de la base de datos con los datos introducidos por el terapeuta. Estos dos componentes se comunican entre sí utilizando tecnologías de Servicios Web RESTful. La arquitectura definida utiliza este enfoque para facilitar su composibilidad con futuros juegos serios desarrollados para el exoesqueleto. Así, la configuración de dichos juegos, como los patrones y los niveles de dificultad, se basará en la información histórica de un paciente concreto.

INDEX

FIGURE INDEX

TABLE INDEX

1

1 INTRODUCTION

1.1 Cerebral Palsy

Cerebral palsy (CP) is a disorder that causes disturbances in a person's muscle tone, motor skills, ability to move, posture, balance, and coordination [1]. It is the most common motor disability in children [2] and affects 2-3 out of 1000 children. [3] Originally, lack of oxygen during childbirth was thought to be one of the main causes of CP, however it has been determined that the origin of this illness is varied. The brain damage that causes CP can happen before or during birth, and even in the course of the first few years after birth [2], and therefore the risk factors of this illness are usually divided depending on the moment the brain damage occurs.

1.1.1 Aetiology, Types and Symptoms

Although cerebral palsy has been described by medical professionals since the 1800s, the risk factors and their complex nature were not studied until the 20th century [4]. The risk factors of CP are varied and categorised as follows: pre-conception, such as malnutrition, drugs or immune system disorders; pre-natal, like maternal bleeding and blood-clotting or toxins; peri-natal, being premature or breech birth; and post-natal and infant period, caused by infections, head trauma or lack of oxygen [5] [6]. Out of all these risk factors, prematurity has the highest incidence of CP, and the severity of the disorders is correlated with pregnancy duration; more premature babies will present more serious afflictions [5].

Cerebral palsy has a varied clinical presentation, and it is therefore important to manage its classification. Due to this variety, there are several ways to group and define the different disorders of CP. According to the Surveillance of Cerebral Palsy in Europe (SCPE) [7] (Table 1), CP has can be divided into four types based on the part or parts of the body affected. Further forms of classification have also been established, such as the Gross Motor Function Classification System (GMFCS) which orders the child's ability to move into five different levels [8] (Table 1). Other forms of classifications exist such as the Manual Ability Classification System (MACS) and the Communication Function Classification system (CFCS) [9].

Table 1 Most common forms of CP classification: SCPE and GMFCS.

Type	**Characteristics**
Surveillance of Cerebral Palsy in Europe	
Spastic	The most common type of CP, affecting around 80% of people who suffer from it. There are different types depending on the body parts that are affected: spastic diplegia when both legs are affected, and the arms are less affected or not affected at all; spastic hemiplegia, when one side of the body is affected (one arm and a leg); spastic quadriplegia, affects all limbs, and often affects hearing, vision and speech [10] [5].
Dyskinetic	This type of CP causes people to have difficulty controlling their movements. It may also affect the face and tongue, so the person might have trouble swallowing. There are two types, dystonic CP when the muscle tension is high, and choreoathetotic CP, when the muscle tension is low. [5]
Ataxic	This type of CP causes difficulties in coordination and balance. A person suffering from this CP has low muscle tension and can't perform fast movements that need control. [10] [5]
Mixed	Mixed CP is when a person has characteristics from more than one type of CP. [10]
Gross Motor Function Classification System	
Level I	The person can walk freely and can perform gross motor functions such as jumping, but speed and coordination are limited. [5] [11]
Level II	The patient may need support for some activities such as climbing stairs and may tire after walking long distances. [5] [11]
Level III	The person needs support equipment such as a hand-held mobility device or a wheelchair for long distances. [5] [11]
Level IV	The person needs a mobility device that is either powered or requires physical assistance from another person but can still move on their own. [5] [11]
Level V	The person cannot move on their own. [5] [11]

1.1.2 Treatments and Therapies

Since cerebral palsy is an umbrella term for a variety of disorders, the treatments for this disease are also varied, depending on the type and age of the person. There are surgical treatments, physical therapy, and medications such as Botulinum toxin or Baclofen [12].

Physical therapy has been proved to benefit individuals with cerebral palsy [13], improving gait and balance. Physical therapy using specially developed games is known for having positive results [14] on trunk control and balance. Using virtual reality games with an exoskeleton also proved to be helpful in improving performance [15].

This project will be centred in the use of an exoskeleton during CP physical therapies. This exoskeleton (Figure 1) has two arms where the patient will introduce their upper limbs. These arms have can be used to assist or resist biceps flexion, and the difficulty of such movement can be increased or decreased.

Figure 1 Exoskeleton used in our therapy sessions

1.2 Objectives and Structure

The aim of the thesis is to create a software architecture capable of managing two microservices: one used to store and handle large amounts of patient and session data, and another used to run an expert system that will be able to recommend a course

of treatment and therapy given the patient's history. To accomplish this, the system would need information about the patient and their rehabilitation process. This data can be recorded using an exoskeleton which can register the movement of the patient and the activity of the muscle. This information is retrieved at extremely high frequencies, generating copious amounts of data, which has to me stored in a manner that is easily retrieved. When these details have been correctly saved, they can then be used by an inference engine to make decisions and help physiotherapists determine a course of action in the rehabilitation process.

Having set the objectives, the project can be divided into two main blocks: the creation of non-relational database used to insert and extract the data, and a rule-based system able to infer and make recommendations about the proper therapy course. Both of these will be wrapped into two microservices. This has been done in case of a future implementation of the application with added information of the rehabilitation games that can be used with the exoskeleton. Although the games have not yet been considered in this project, we wanted to create a flexible architecture that can be adapted to future goals. Since the aim is to create a software architecture, to test the system we added another goal: creating an intuitive user interface. This way the potential of the application can be clearly seen, and it will be easier to use.

2 BACKGROUND AND STATE OF THE ART

As explained before, the project is divided into two main microservices, each of which requires choosing between the different types of systems and software available to carry out the intended purposes. This section explains the main options of the three major types of technology used in the project: a database, an expert system, and Web Services.

2.1 Databases

There are two types of databases: relational and non-relational databases. Relational databases are the more traditional model, and until recently, the most widely used. Although each of these models have different forms of storing data, they all differ in in the flexibility and structure. Relational databases have a structured schema and are much more rigid and consistent than non-relational.

2.1.1 Relational Databases

Relational or SQL databases have been the most popular choice until recently. This type of databases uses transactions, which give them a property known as "ACID-ity". This stands for Atomicity, Consistency, Isolation and Durability, which means that the information is only committed if it is entirely correct and filled [16].

These databases are a collection of information organised in tables, and further divided into columns [17]. Each row contains an instance of data for each column. These rows can be accessed using unique keys, and referential integrity is maintained thanks to the ACID properties.

However, these properties are what also make relational databases less desirable when working with large amounts of data, as they require a transaction manager, and due to their atomicity, each transaction has to be completed before another one can be committed.

2.1.2 Non-relational Databases

There are four main types of non-relational databases: document-oriented, wide-column, key-value, and graph databases. We will study the first two, as they are the most commonly used and versatile.

Figure 2 Representation of document-oriented databases (left) and wide-column based databases (right).

Document-oriented databases are currently one of the most widely used NoSQL database types, some examples include MongoDB and CouchDB. This type of database stores documents (Figure 2). These documents are schema-less and are usually written in JSON format. This has an unmistakable advantage over relational databases, as they can store in a single document what a relational database would store in several tables, requiring several *joins* to query the. This type of database also provides horizontal scaling, which means that a node can be added when needed (Figure 3).

Wide-column databases are conceptually the most comparable to traditional databases (Figure 2) [18]. Data is organised into columns, which are then grouped together, and these groups of columns are then linked by a row containing the primary key. Wide-column databases scale both horizontally and vertically. Vertical scaling consists in adding more resources and power, such as CPU and RAM (Figure 3).

This type of databases are particularly useful when handling Big Data, as they allow for scalability and availability, allowing for large volumes of data to be stored as well as quick access to it. They are schema-less allowing the storage of unstructured data, as well as semi-structured and structured, making them more flexible.

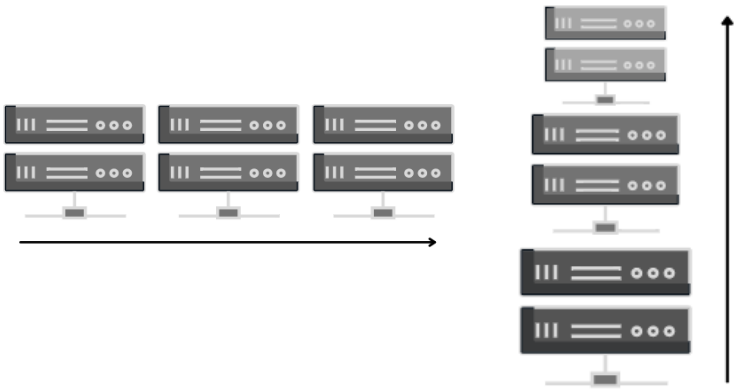


Figure 3 Horizontal scaling (left): handles the growing amount of data by adding more machines. Vertical scaling (right): adds more resources to a single node.

2.2 Expert Systems

An expert system is a program that applies knowledge to the problem-solving process of a specific field [19]. There are several types of expert systems [20]. All expert systems are composed by an inference engine, and a representation of the knowledge, known as the knowledge base of the system.

2.2.1 Fuzzy-Logic Systems

These systems use fuzzy logic, which describes knowledge as a sliding scale. Since they don't used fixed numerical values, they are used to describe vague terms and differentiate between members and non-members of a class using multi-valued logic, making them useful when dealing with uncertainty [21]. The rules in these expert systems usually follow this form: `if X is low and Y is high, then Z = medium.`

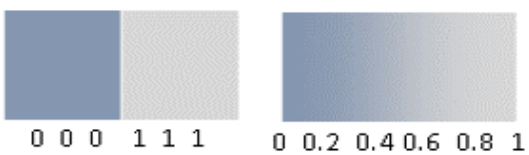


Figure 4 Representation of Boolean Logic (left) and Fuzzy or Multi-valued Logic (right)

2.2.2 Frame-Based Systems

These systems use frames or data structures to capture and represent knowledge [22]. These frames have slots which describe attributes associated with the frame, and facets provide extended knowledge on said attributes (Figure 5). They provide a natural way for the representation of knowledge. They are an application of object-oriented programming for expert systems.

User	
Slots	Facets
Name	(Type:String)
DOB	(Type:Date, Range:01/01/1900 ..31/12/2021)
Height	(Type:Real, Units:meters)

Figure 5 Example of a Frame

2.2.3 Hybrid Neural Systems

These systems store knowledge as weights in neurons imitating human intelligence [23]. They can either combine neural networks with rules, which have a neural knowledge base where knowledge is stored as weights in neurons, or with fuzzy logic (Neuro-Fuzzy Systems), which combine the parallel computation of neural networks with human-like knowledge representation of fuzzy systems.

2.2.4 Rule-Based Systems

In rule-based expert systems knowledge is expressed as a set of rules which are divided into two: the antecedent and the consequent [20]. These are represented as IF (condition) and THEN (conclusion) statements, i.e. `IF patient has Triceps Atrophy THEN change exoskeleton configuration.` Figure 6 shows the main elements of expert systems previously mentioned (inference engine and knowledge representation), applied to rule-based systems.

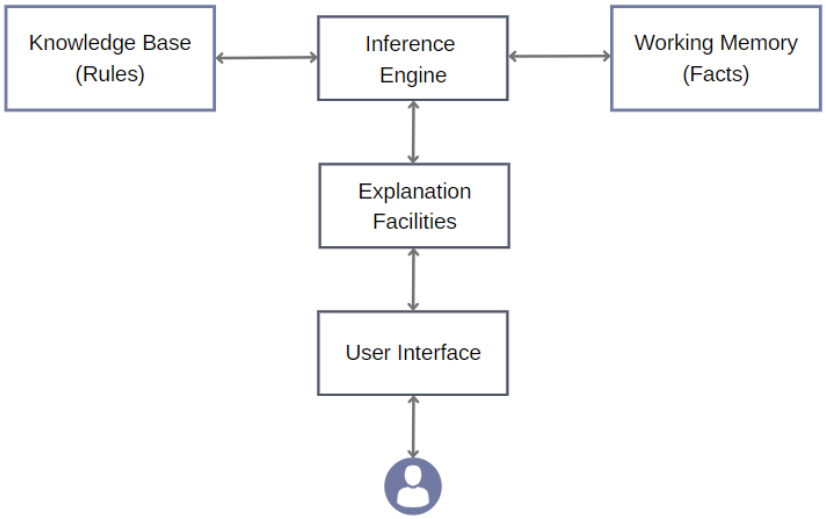


Figure 6 Structure of a Rule-Based System

2.3 Web Services

A web service is a piece of software or application that can communicate with each other using standardised protocols throughout a network [24]. The communication is usually carried out between a client and a server. The client sends requests to the server, sometimes including data so that the server can process the request. The server then generates a response with another set of data and sends it back to the client. There are many types of web services, varying in which protocol they use to communicate and how the data is transferred between the server and the client.

We refer to as microservices as the approach of application development using a suite of small services. This way, each function of an application can be built independently. These small services are loosely coupled, which reduce the dependencies between them.

Today, there are two architectural styles for the creation of web services: Service Oriented Architecture, based on the WS-* protocols and used by traditional Web Services, and Resource Oriented Architecture, based on the HTTP protocol and used by RESTful Web Services.

2.3.1 Traditional Web Services

These web services are based on WS-* protocols and use SOAP messaging to communicate and exchange data between the client and the server. These messages are XML documents that have different elements: an *envelope* that identifies the message as a SOAP message, a *header* with header information and the *body* with call and response information (Figure 7).

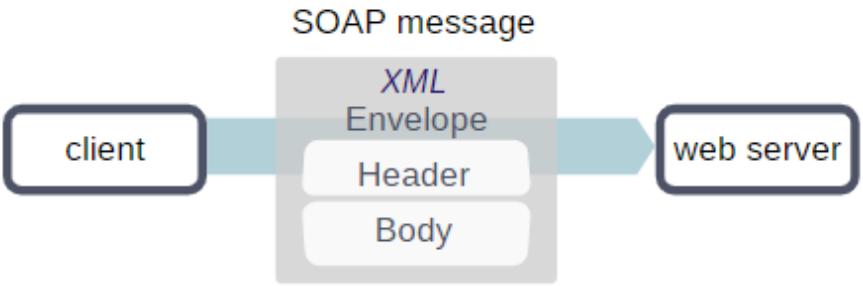


Figure 7 SOAP communication

SOAP is used with another set of complex protocols to complete the Service Oriented Architecture (Figure 8). These protocols are WSDL (Web Services Description Language) and UDDI (Universal Description Discovery and Integration). They are used to communicate the Service Provider (server) and Consumer (client) to the Service Registry. The Registry provides controlled access and reference information about the software components that are available and how they are connected to each other.

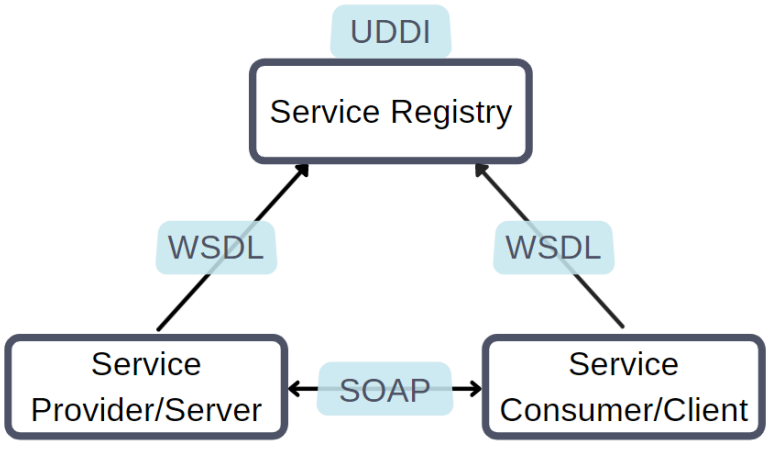


Figure 8 SOA Architecture

2.3.2 RESTful Web Services

RESTful Web Services follow an architectural style based on the Web [25]. Roy Fielding first described the REST (Representational State Transfer) architectural style and identified the architectural elements: resources, resource identifiers, representations of the resource used as a response to a request, and messages through which the resources are manipulated (all resources provide an uniform interface, which relies on the HTTP protocol when using the Web as underlying platform) [26]. REST uses a client-server approach and communicates sending messages which can be written in XML, JSON and other forms (Figure 9).

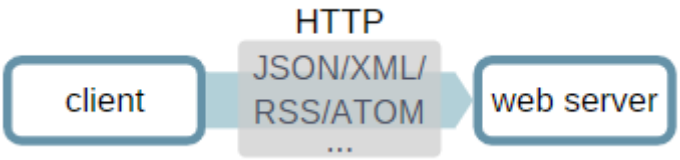


Figure 9 Communication in RESTful Web Services

There are also several principles that require the correct design of a RESTful Web Service [25]:

- Addressability, which means that resources are accessed using a Universal Resource Identifier (URI). This means that an identification and representation of a resource is needed. A resource is a key element of the application (Patient, Session, etc.) that must be unique and identifiable in a standardised manner. This is done using URIs, such as `/patients or /patients/:id.` We refer to *resource representation* as the media type the resource is represented in the web at a certain time, such as HTML or JSON.
- Statelessness, which means that every transaction must be independent, as all the data required to process the request is contained on said request. Representational State Transfer implies that the client and server manipulate the state by sending representations in response to the requests.

- Uniform Interface, which requires the use of standard HTTP methods such as GET, POST, PUT and DELETE. These allow the obtention of the resource representation, the creation of a resource, the updating of a resource, or the elimination of a resource respectively.
- Connectedness, which refers to the property which allows a resource to be reached through another resource by sending links and forms in its representations. This was described by Fielding as *Hypermedia as the Engine of the Application State* (HATEOAS).

3 ANALYSIS AND REQUIREMENTS

Before we can design the application, an exhaustive analysis of the domain must be done to identify all the resources and processes it may need and use. Since the database and the expert system will be mostly using the data provided by the exoskeleton, we will begin by studying it and how it works, plus the data its sensors capture. Some of the data will be given by the therapist managing the exoskeleton. Based on this, we will define the resources and processes that the app will be able to carry out.

3.1 Domain Analysis

3.1.1 The Exoskeleton

The exoskeleton consists of two arms like the one shown in Figure 10. These arms have bolts where they elastics can be hooked on. The more elastics are added, and depending on the configuration, flexion becomes either much easier or harder. In addition to the exoskeleton, the patients wear an *mDurance* sensors that are able to capture the EMGs and ROMs.

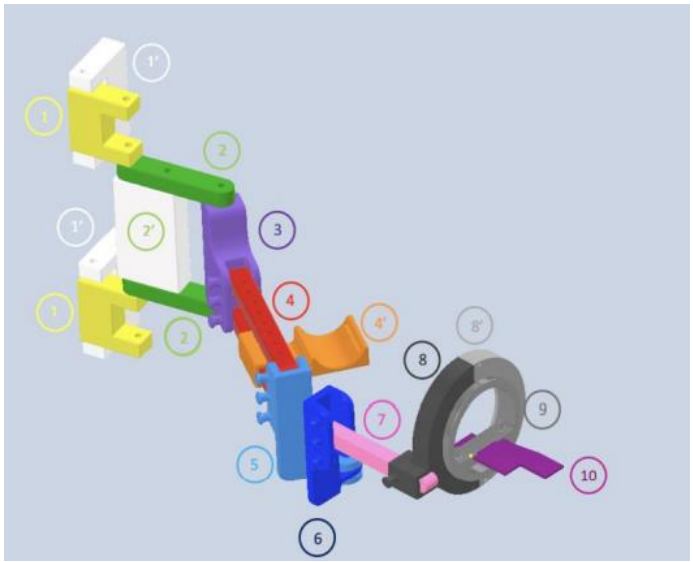


Figure 10 The exoskeleton arm and its parts.

The exoskeleton has essentially two "modes": assisting biceps flexion and resisting it. These configurations are changed by changing the positions of the elastics in the exoskeleton. To assist biceps flexion (biceps contracts), they must be placed as it is shown on the left configuration of Figure 10, to resist biceps flexion (essentially assist triceps extension/contraction) they must be placed as they are on the right configuration of Figure 11.

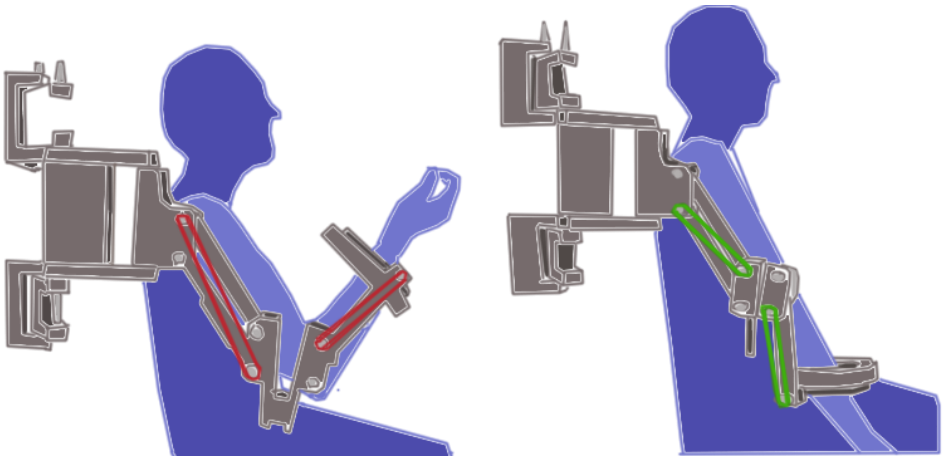


Figure 11 Elastic placement to assist biceps flexion (red) and resist biceps flexion (green).

3.1.2 mDurance sensor and application

Figure 12 mDurance sensor [27]

This sensor is a 30 grams device that is able to record EMG and ROM data at a frequency of 1kHz. Two sensors were used with the exoskeleton, recording the activity of four muscles. However, only the values of the biceps and triceps were finally used since they were the ones that provided the most information about the influence of the exoskeleton [28]. Throughout the paper the four values may be referenced, but ultimately only two (EMG1 and EMG2) are used. These sensors can be paired to Android devices and their data can be seen through the mDurance app. This app not only stores the information recorded by the sensors, but it also allows the creation of

patient profiles, where information such as name, weight and date of birth are saved, as shown in Figure 11.

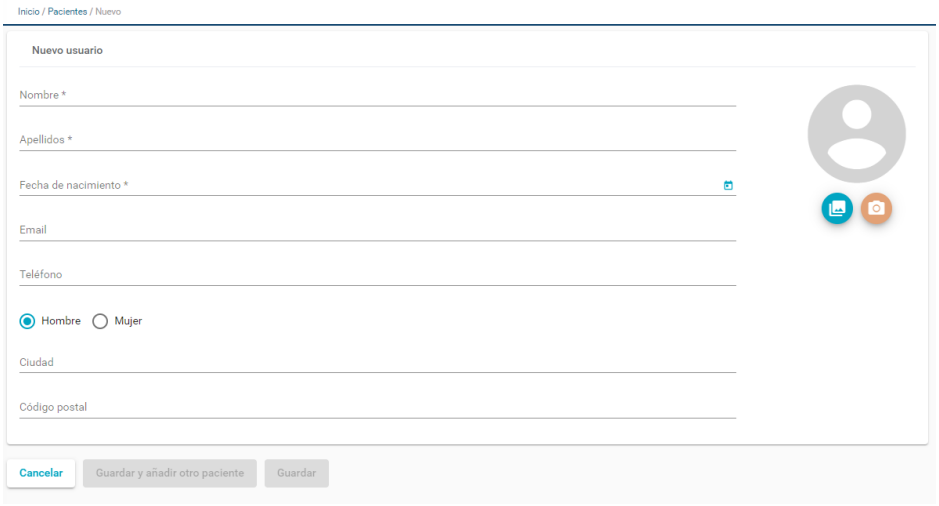


Figure 13 Patient creation in the *mDurance* app

In summary, the sensor and the exoskeleton provide information that will be later used to draw conclusions and predict future therapies. These data are the number of elastics, whether they are assisting or resisting biceps flexion, and patient information, which may include a diagnosis. These will help determine the resources of the application in the next section.

3.1.3 Data analysis: Entities

We refer as entities groups of related data, or objects on which we want to perform operations. These entities can be related to each other. These can be represented as entities with attributes that describe them. To determine our entities, we must first look at the data we can obtain, the information we want to produce, and how it can be divided logically into groups.

As explained in the previous section, the app allows to introduce all the data of the patient. This is a clear entity, as it is related to the other information we pick up with the sensors and the exoskeleton, yet it has its own attributes that are separate to the rest. This will be represented by the entity *Patient*.

The following entities are not as clear as *Patient*. We have raw data that we record with the sensors, but this data is ultimately not what we will use to predict future therapies. It is used, however, to calculate useful values that can be used to describe how the muscles work and therefore predict future rehabilitation processes. Between these two groups of data there is a clear division: data that is recorded and data that is calculated. The information from the sensors will be represented by the entity *Raw Data*, whilst the calculated information will be called *Session*, representing the useful data that can be extracted from a therapy session. The entities are related to each other as shown in Figure 14. Each set of *Raw Data* is associated to a single *Session*, and each *Session* is related to a patient.

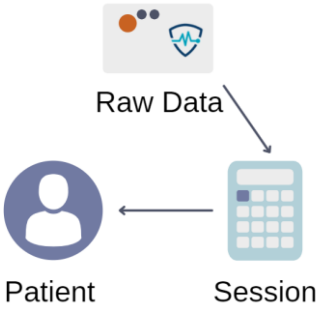


Figure 14 Resource associations.

As explained before, *Raw Data* consists of the EMGs, and the ROMs recorded by the sensors. *Session* however has a different set of data that can be achieved using all the *Raw Data* associated with that *Session*. This data is [28]:

- Recruitment of muscle fibres (RMS) (µV): represents neuromuscular signals recorded by EMG sensors with RMS smoothing. It takes the average of the number of fibres that cause the muscle to act.
- Maximum Voluntary Contraction (MVC) (µV): it is the highest peak in the RMS smoothed signal, when the most muscles fibres are acting.
- Muscle Activity (%): how active the muscle is compared to others (difference in contractile activity between synergist and antagonist muscles).

How these values are calculated will be explained in the section 5 of the paper.

3.1.4 Statistical study of Relevant Parameters

As it has been previously explained, this project will have an expert system used to make predictions for future rehabilitation processes. The knowledge base of the Expert System requires certain parameters to be able to reach a conclusion about future therapy sessions. Therefore, we must determine which data is relevant in the muscle behaviour to be able predict future behaviours and therapies, so that the parameters fed into the Expert System are useful when reaching a conclusion.

The parameters we have to consider are those that are not measured by the sensors since these are the ones that measure muscle activity. Those would be the patient, and the number of elastics used. At first glance, it may seem that the patients' physical characteristics, such as height and weight, may influence how the muscle acts.

A correlation analysis using IBM's SPSS using Spearman (Figure 15) and Pearson (Figure 16) was carried out. The samples used were the results of the testing carried out by Laura Blanco [28] using the exoskeleton with healthy children. These were used as we assumed normality, since we did not have many samples. Using Spearman we wanted to check if any of the physical characteristics were related to the muscle activity (these are labelled as biceps or triceps in Figure 15), and therefore if they influenced it at all. The results show that the muscle activity was not dependant of these characteristics. The Pearson correlation shows similar results, showing that there is no linear relationship between the physical characteristics and the muscle activity. Therefore, after this statistical study, we established that the physical traits of the patients did not influence how the muscle behaved and could not be used as parameters to estimate and recommend future treatments.

		peso	altura	beceps0	triceps0	bicess5asis	triceps5asis	Biceps5resis	triceps5resis
peso	Coeficiente de correlación	1,000	,832**	-,126	,319	,341	-,480*	-,234	-,042
	Sig. (bilateral)	.	,000	,606	,183	,153	,037	,335	,863
	N	19	19	19	19	19	19	19	19
altura	Coeficiente de correlación	,832**	1,000	-,080	,173	,279	-,433	-,124	,076
	Sig. (bilateral)	,000	.	,744	,480	,247	,064	,613	,758
	N	19	19	19	19	19	19	19	19
beceps0	Coeficiente de correlación	-,126	-,080	1,000	-,439	,688**	,144	,619**	-,177
	Sig. (bilateral)	,606	,744	.	,060	,001	,557	,005	,468
	N	19	19	19	19	19	19	19	19
triceps0	Coeficiente de correlación	,319	,173	-,439	1,000	-,205	,225	-,170	,345
	Sig. (bilateral)	,183	,480	,060	.	,399	,355	,486	,148
	N	19	19	19	19	19	19	19	19
bicess5asis	Coeficiente de correlación	,341	,279	,688**	-,205	1,000	-,363	,411	-,189
	Sig. (bilateral)	,153	,247	,001	,399	.	,126	,081	,439
	N	19	19	19	19	19	19	19	19
triceps5asis	Coeficiente de correlación	-,480*	-,433	,144	,225	-,363	1,000	-,079	,271
	Sig. (bilateral)	,037	,064	,557	,355	,126	.	,748	,261
	N	19	19	19	19	19	19	19	19
Biceps5resis	Coeficiente de correlación	-,234	-,124	,619**	-,170	,411	-,079	1,000	-,230
	Sig. (bilateral)	,335	,613	,005	,486	,081	,748	.	,344
	N	19	19	19	19	19	19	19	19
triceps5resis	Coeficiente de correlación	-,042	,076	-,177	,345	-,189	,271	-,230	1,000
	Sig. (bilateral)	,863	,758	,468	,148	,439	,261	,344	.
	N	19	19	19	19	19	19	19	19

Figure 15 Correlation analysis using Spearman

		peso	altura	beceps0	triceps0	bicess5asis	triceps5asis	Biceps5resis	triceps5resis
peso	Correlación de Pearson	1	,934**	,001	,246	,371	-,495*	-,065	-,012
	Sig. (bilateral)		,000	,996	,310	,118	,031	,792	,962
	N	19	19	19	19	19	19	19	19
altura	Correlación de Pearson	,934**	1	-,065	,150	,269	-,482*	-,114	-,023
	Sig. (bilateral)	,000		,791	,541	,266	,036	,642	,925
	N	19	19	19	19	19	19	19	19
beceps0	Correlación de Pearson	,001	-,065	1	-,408	,676**	,218	,736**	-,219
	Sig. (bilateral)	,996	,791		,083	,001	,370	,000	,368
	N	19	19	19	19	19	19	19	19
triceps0	Correlación de Pearson	,246	,150	-,408	1	-,173	,190	-,234	,336
	Sig. (bilateral)	,310	,541	,083		,478	,435	,336	,159
	N	19	19	19	19	19	19	19	19
bicess5asis	Correlación de Pearson	,371	,269	,676**	-,173	1	-,367	,482*	-,157
	Sig. (bilateral)	,118	,266	,001	,478		,122	,037	,521
	N	19	19	19	19	19	19	19	19
triceps5asis	Correlación de Pearson	-,495*	-,482*	,218	,190	-,367	1	,091	,255
	Sig. (bilateral)	,031	,036	,370	,435	,122		,712	,291
	N	19	19	19	19	19	19	19	19
Biceps5resis	Correlación de Pearson	-,065	-,114	,736**	-,234	,482*	,091	1	-,335
	Sig. (bilateral)	,792	,642	,000	,336	,037	,712		,161
	N	19	19	19	19	19	19	19	19
triceps5resis	Correlación de Pearson	-,012	-,023	-,219	,336	-,157	,255	-,335	1
	Sig. (bilateral)	,962	,925	,368	,159	,521	,291	,161	
	N	19	19	19	19	19	19	19	19

Figure 16 Correlation analysis using Pearson

Unfortunately, the data that we had to make the analysis was quite limited, as it didn't have many samples. However, to be able to produce results, it was decided that we could use the muscle activity of the samples as a point of reference for the predictions. These samples are taken from healthy children, and could therefore be used to compare it to the muscle activity of children with CP. The closer the muscle activity of the children with CP are to those taken from the samples, the better. Another problem that arose was due to the fact that the muscle activity of our set of samples was recorded with zero and five elastics, which meant that we had to estimate the values in between. The figures below show how these values were estimated assuming linearity using Excel.

ID	BicepsExoOnly	Biceps1Assis	Biceps2Assis	Biceps3Assis	Biceps4Assis	Biceps5Assis
19	63.41	60.98	58.55	56.12	53.69	51.26
18	42.5	41.42	40.34	39.26	38.18	37.1
17	46.17	44.088	42.006	39.924	37.842	35.76
16	69.39	62.534	55.678	48.822	41.966	35.11
15	53.81	50.78	47.75	44.72	41.69	38.66
14	47.44	45.926	44.412	42.898	41.384	39.87
13	55.35	51.894	48.438	44.982	41.526	38.07
12	36.81	34.234	31.658	29.082	26.506	23.93
11	53.82	49.504	45.188	40.872	36.556	32.24
10	48.4	45.578	42.756	39.934	37.112	34.29
9	46.58	46.308	46.036	45.764	45.492	45.22
8	29.11	28.304	27.498	26.692	25.886	25.08
7	41.78	39.102	36.424	33.746	31.068	28.39
6	45.44	43.436	41.432	39.428	37.424	35.42
5	39.38	36.364	33.348	30.332	27.316	24.3
4	61.02	57.762	54.504	51.246	47.988	44.73
3	35.75	34.696	33.642	32.588	31.534	30.48
2	52.2	48.162	44.124	40.086	36.048	32.01
1	41.99	39.252	36.514	33.776	31.038	28.3
Mean	46.90732346	44.39995468	41.86442665	39.29514946	36.68450173	34.02160563
# elsatics	0	1	2	3	4	5
Median of Mean	40.57978806	40.57978806	40.57978806	40.57978806	40.57978806	40.57978806
SD	4.818701073					
Mean of Mean	40.28785238	40.28785238	40.28785238	40.28785238	40.28785238	40.28785238

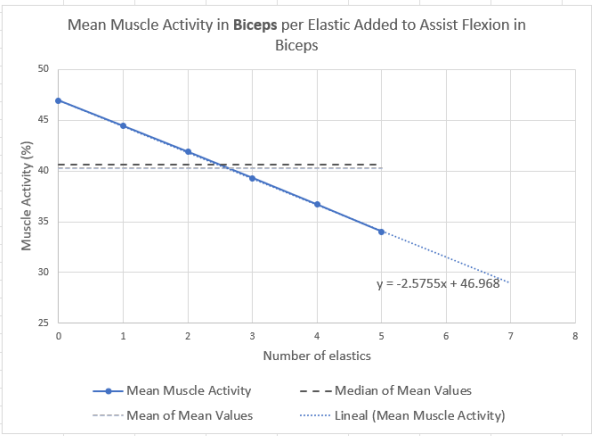


Figure 17 Estimation of Biceps Muscle Activity when the elastics assist flexion

ID	BicepsExoOnly	Biceps1Resis	Biceps2Resis	Biceps3Resis	Biceps4Resis	Biceps5Resis
19	63.41	62.944	62.478	62.012	61.546	61.08
18	42.5	44.302	46.104	47.906	49.708	51.51
17	46.17	47.218	48.266	49.314	50.362	51.41
16	69.39	69.374	69.358	69.342	69.326	69.31
15	53.81	52.912	52.014	51.116	50.218	49.32
14	47.44	49.786	52.132	54.478	56.824	59.17
13	55.35	54.934	54.518	54.102	53.686	53.27
12	36.81	37.928	39.046	40.164	41.282	42.4
11	53.82	55.04	56.26	57.48	58.7	59.92
10	48.4	48.266	48.132	47.998	47.864	47.73
9	46.58	47.258	47.936	48.614	49.292	49.97
8	29.11	30.84	32.57	34.3	36.03	37.76
7	41.78	45.286	48.792	52.298	55.804	59.31
6	45.44	48.376	51.312	54.248	57.184	60.12
5	39.38	39.12	38.86	38.6	38.34	38.08
4	61.02	59.958	58.896	57.834	56.772	55.71
3	35.75	37.28	38.81	40.34	41.87	43.4
2	52.2	52.346	52.492	52.638	52.784	52.93
1	41.99	44.998	48.006	51.014	54.022	57.03
Mean	46.90732346	47.99338659	49.03814318	50.04479826	51.01603971	51.95414497
# elastics	0	1	2	3	4	5
Median of Mean	49.54147072	49.54147072	49.54147072	49.54147072	49.54147072	49.54147072
SD	1.888005361					
Mean of mean	49.46222616	49.46222616	49.46222616	49.46222616	49.46222616	49.46222616

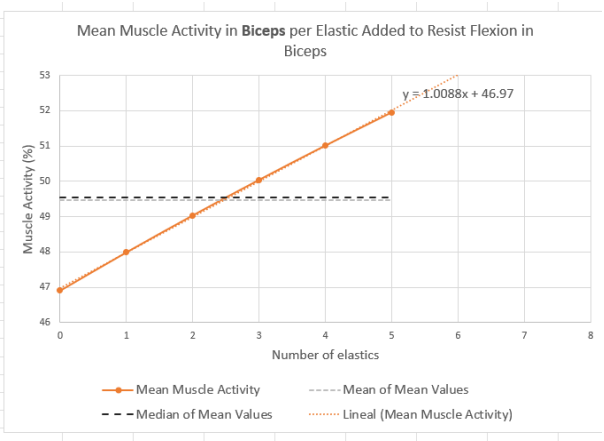


Figure 18 Estimation of Biceps Muscular Activity when elastics resist flexion

ID	TricepsExoOnly	Triceps1Assis	Triceps2Assis	Triceps3Assis	Triceps4Assis	Triceps5Assis
19	14.91	21.492	28.074	34.656	41.238	47.82
18	19.74	23.542	27.344	31.146	34.948	38.75
17	11.64	15.72	19.8	23.88	27.96	32.04
16	9.28	14.218	19.156	24.094	29.032	33.97
15	15.07	16.856	18.642	20.428	22.214	24
14	18.18	21.988	25.796	29.604	33.412	37.22
13	9.21	15.236	21.262	27.288	33.314	39.34
12	9.54	11.004	12.468	13.932	15.396	16.86
11	10.13	14.912	19.694	24.476	29.258	34.04
10	11.41	14.644	17.878	21.112	24.346	27.58
9	8.11	11.296	14.482	17.668	20.854	24.04
8	14.98	19.81	24.64	29.47	34.3	39.13
7	10.47	13.856	17.242	20.628	24.014	27.4
6	13.06	16.198	19.336	22.474	25.612	28.75
5	18.05	21.728	25.406	29.084	32.762	36.44
4	8.94	14.524	20.108	25.692	31.276	36.86
3	16.87	17.81	18.75	19.69	20.63	21.57
2	11.78	15.928	20.076	24.224	28.372	32.52
1	19.05	19.788	20.526	21.264	22.002	22.74
Mean	12.67057258	16.50603034	20.16641789	23.72882603	27.22765539	30.68163041
# elsatics	0	1	2	3	4	5
Median of Mean	21.94762196	21.94762196	21.94762196	21.94762196	21.94762196	21.94762196
SD	6.724564725					
Mean of Mean	20.91086969	20.91086969	20.91086969	20.91086969	20.91086969	20.91086969

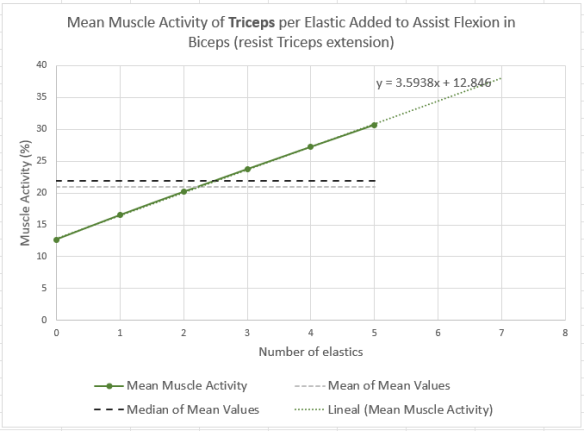


Figure 19 Estimation of Triceps Muscular Activity when elastics assist flexion

ID	TricepsExoOnly	Tricpes1Resis	Triceps2Resis	Triceps3Resis	Triceps4Resis	Triceps5Resis
19	14.91	15.094	15.278	15.462	15.646	15.83
18	19.74	18.574	17.408	16.242	15.076	13.91
17	11.64	11.028	10.416	9.804	9.192	8.58
16	9.28	9.534	9.788	10.042	10.296	10.55
15	15.07	14.796	14.522	14.248	13.974	13.7
14	18.18	16.222	14.264	12.306	10.348	8.39
13	9.21	10.452	11.694	12.936	14.178	15.42
12	9.54	9.742	9.944	10.146	10.348	10.55
11	10.13	9.952	9.774	9.596	9.418	9.24
10	11.41	12.542	13.674	14.806	15.938	17.07
9	8.11	7.84	7.57	7.3	7.03	6.76
8	14.98	15.73	16.48	17.23	17.98	18.73
7	10.47	10.576	10.682	10.788	10.894	11
6	13.06	13.078	13.096	13.114	13.132	13.15
5	18.05	16.904	15.758	14.612	13.466	12.32
4	8.94	8.998	9.056	9.114	9.172	9.23
3	16.87	16.29	15.71	15.13	14.55	13.97
2	11.78	11.436	11.092	10.748	10.404	10.06
1	19.05	17.746	16.442	15.138	13.834	12.53
Mean	12.67057258	12.56973124	12.42915838	12.24847798	12.02524697	11.75415011
# elsatics	0	1	2	3	4	5
Median of Mean	12.33881818	12.33881818	12.33881818	12.33881818	12.33881818	12.33881818
SD	0.346727588					
Mean of Mean	12.27878004	12.27878004	12.27878004	12.27878004	12.27878004	12.27878004

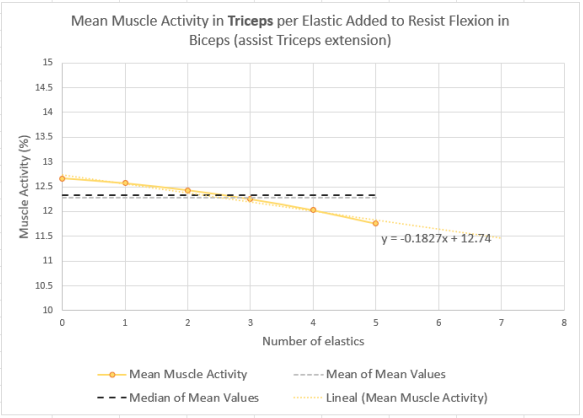


Figure 20 Estimation of Triceps Muscular Activity when elastics resist flexion

Figure 17 and Figure 18 show the muscle activity for the biceps when the elastics are assisting and resisting flexion. Since we had the measurements for the muscle activity with zero and five elastics, the muscle activity for 1 through 4 elastics was estimated using these values and assuming linearity. These values were then used to estimate a function which would give the muscle activity given a certain number of elastics. The same process was done with the triceps muscle activity, shown in Figure 19 and Figure 20.

These linear values, and an added margin, will indicate if the values recorded during sessions are on the right track, or if they are too low or too high given the number of elastics used. The linear functions obtained will be used to feed values to the expert system when comparing them.

3.1.5 Processes

There are different functions and therefore processes the application provides. These will be explained more thoroughly hereafter. To aid the explanations of the processes, they are represented working together in Figure 21.

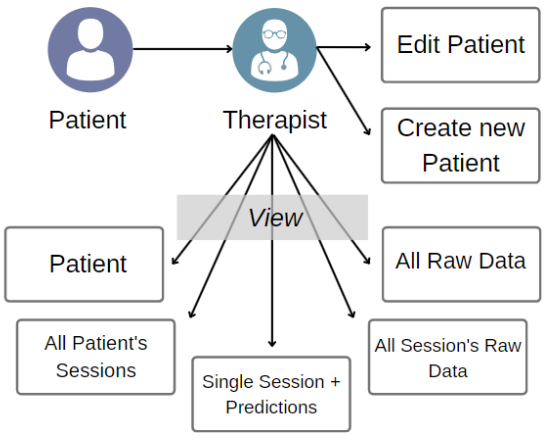


Figure 21 All actions the therapist can perform with the app.

The whole application will be managed by the therapist. They will be able to insert a new patient, providing their data that will be later stored in the database. This patient information could be later viewed, edited, or deleted. The therapist may also see the raw data, either by session or the whole set that is stored in the database. All the sessions associated with a patient will be able to be viewed, as well as a single session of the therapist choosing, which will be the one that is used to make the predictions.

3.2 Requirements

The previous sections that describe the state of the art and the domain analysis for this project allow us to establish a set of requirements that will allow to guide the design and implementation of the application to manage our objectives.

The first requirement we have to complete is the storage of our data. As it has been previously explained, the sensors capture large amounts of data, recording EMGs at a frequency of 1kHz. This means that the database we choose must easily store and adapt to large amounts of data and should also be flexible as not all patients may have the same number of sessions, and some of these sessions may be longer than others.

As indicated by the statistical analysis, we must also use and provide muscle activity measurements. This value has to be calculated from the EMG recordings of a session, therefore the creation of an algorithm capable of calculating said value is required.

An expert system which can be fed with various parameters and conditions and then reach a conclusion is also needed. The expert system must be the most efficient and intuitive with the knowledge we provide.

These requirements will be done using two RESTful Web Services, which will allow a better integration with other software in the future. One will manage the database and the other the Expert System, creating an architecture that is ready to be integrated in the future with the rehabilitation games that are used alongside the exoskeleton.

We must also provide a user interface where the therapist can execute all the processes described previously in this section. This user interface has to be visually compelling so that results can be correctly interpreted.

4 ARCHITECTURE AND DESIGN

4.1 Proposed Architecture

The proposed architecture for the application is composed of two microservices in the backend of the application, and a user interface making up the frontend of the system, as shown in Figure 22.

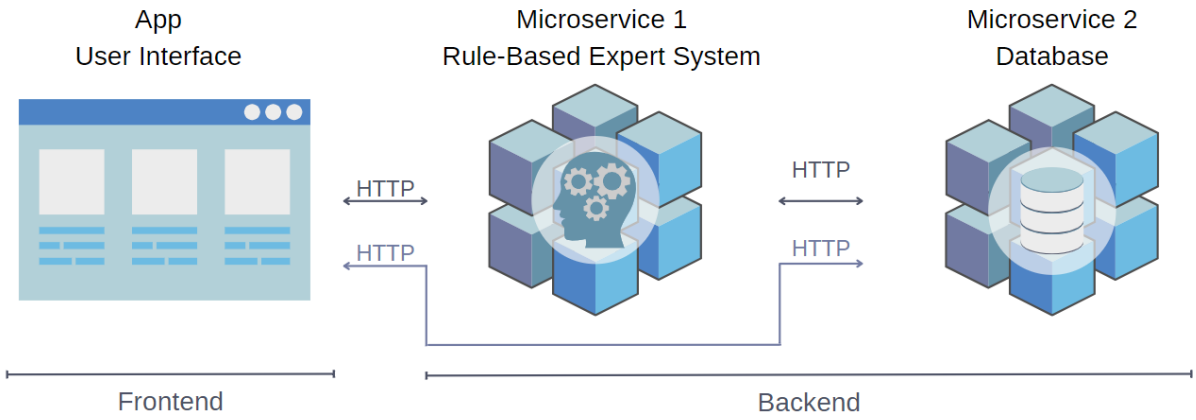


Figure 22 Architecture design of the entire application

As shown on the figure above, the backend of the application is divided into two microservices. The first microservice consists of a rule-based system, as it is the most effective to carry out the requirements. This microservice is in charge of making decisions based on the input of the user, the data stored in the database and the knowledge base of the expert system. It does so by using a set IF-THEN rules as explained previously. The second microservice consists of a NoSQL database, as we need to store large amounts of data in a flexible manner. It will store all the users, session and raw data captured. Since Microservice 1 (MS1) needs to communicate with Microservice 2 (MS2) to provide conclusions, HTTP requests will be sent between microservices to obtain the necessary information to provide the results needed.

The frontend of the application is what the user sees and interacts with. It will communicate with the backend part of the application using HTTP requests.

Figure 22 shows how the communication between all components works. The user will use the UI to obtain a set of data or estimate the next course of treatment.

When they input all the data, they fill out a template with certain fields. The information from the template will be used to decide which HTTP request will be sent to the backend part of the app. Then, depending on what the user requires, it will either communicate directly with MS2 to obtain the data that it need directly from the database, or it will communicate with MS1, obtaining the information from that microservice. MS1 will then acquire whatever information it needs from MS2 to then provide the complete request to the user.

4.2 Backend Design

The backend design is mainly the development of two RESTful Web Services that will follow the architectural style explained in section 2.3.2. Both microservices will be explained separately as one is dedicated to the expert system, and another to the database. However, both have the same components such as the resources and the corresponding URIs (Uniform Resource Identifier). We will begin by explaining MS2, as it better describes resources that are also used in MS1.

4.2.1 Resource Definition

Based on the analysis done on section 3, the following resources were identified:

- **Patients:** this resource allows the access and carrying out of any operations in relation to the whole set of patients.
- **Patient:** resource that manages all the operation in relation to a single patient.
- **All Raw Data:** this resource permits the management of the raw data and carrying out any operations that relate to a set of raw data.
- **Data:** this resource is used to manage certain raw data given its session ID.
- **Sessions:** this resource manages the operations for all the sessions of a single patient.
- **Session:** this resource refers to a single session of a patient. It manages all the operations related to single sessions.

All these resources are used by MS2, and MS1 uses them all except those referring to the Raw Data.

4.2.2 Microservice 2: Database

We will firstly explain the database and the information it holds, and then the RESTful API that will be used for the access of the database.

4.2.2.1 The Database

The previously described resources will have to be stored in the database in an intuitive manner. Although they are interconnected between them, there are three clear divisions: patients, sessions, and raw data. These will be the tables in the database, and each will contain certain fields, some of which were mentioned in the analysis and requirements section and are shown in Figure 23.

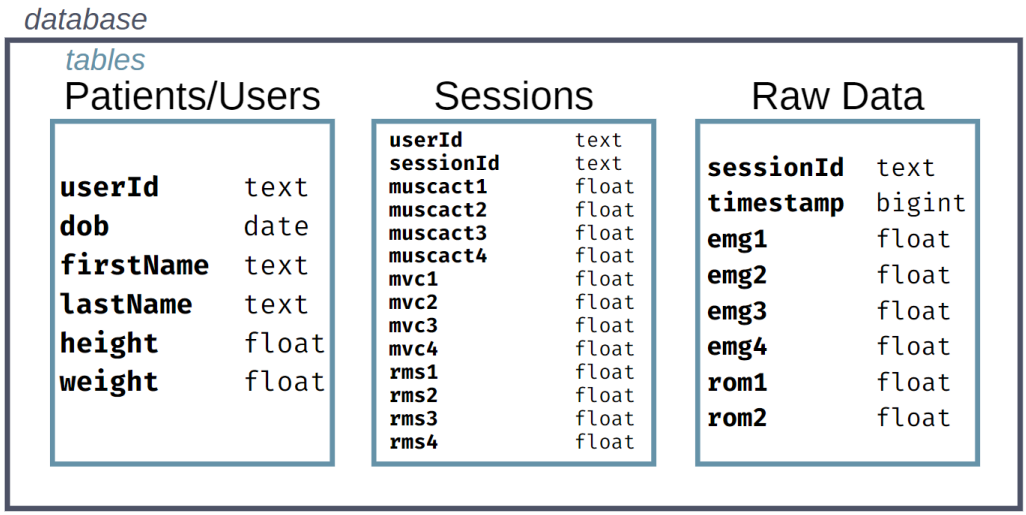


Figure 23 Tables and their fields in the database

The *Patients* table stores the basic information of a patient, such as first name, last name, and date of birth. The Raw Data table stores the EMGs and the ROMs which both have a sampling frequency of 1kHz, the timestamp at which they were recorded and the ID of the session they belong to. The Session table stores the useful information for the process of the therapy. These values are calculated using those in the raw data, as explained previously in the Analysis and Requirements section.

Figure 23 shows how the database will be organised into three different tables, each representing a resource.

4.2.2.2 Identification of Resources

In this section all the URIs used in the API REST for the database will be explained. Some of these URIs refer to something we called a "template". These are fields that the user can fill out with the necessary information to carry out the requests.

- **/patients:** receives all HTTP requests that manage the whole existing list of patients, including the addition of a patient to the list.
- **/patients/:userID:** accesses the patient information given their id.
- **/data:** handles all HTTP requests needed to manage all the raw data in the database.
- **/data/:sessID:** request that handles the raw data given its session ID.
- **/patients/:userID/sessions:** request for the access of all sessions of a patient given their ID.
- **/patients/:userID/sessions/:sessID:** request that manages a single session from a patient given their IDs.
- **/patients/sessions/data:** receives all HTTP requests that require the insertion of data in a session.

Because we want to integrate our application with an HTML frontend, we will offer resources we called "template", which will provide an arrangement of fields to perform certain requests (e.g. POST) on the API.

- **/patients/template:** receives all HTTP requests that require a template with fields that allow the user to interact with the patient data.
- **/patients/template/:userID:** receives all HTTP requests required to the template necessary to edit a patient.
- **/data/template:** receives all HTTP requests that require a template to interact with the data.
- **/patients/sessions/template:** receives all HTTP requests that require a template to interact with session data.

4.2.2.3 RESTful API resource design

For every resource previously determined, a table has been created to further explain how each one works, and the responses generated for each request. For the sake of simplicity, only the HTTP methods that are allowed for a certain URI will be included, i.e., if the table does not include a POST method, it means that said method is not allowed or necessary.

4.2.2.3.1 Patients

Table 2 HTTP methods for the URI /patients

URI: /patients				
HTTP method	Request Body	Function	Response Code	Response Body
GET	-	Lists all patients in the database	200 – OK	Shows list of patients
			404 – Not Found	-
			400 – Bad Request	-
			500 – Internal Server Error	-
POST	Patient information: name, id, height, etc.	Sends the information from the filled-out template fields to the database so	201 – Created	Shows message stating it was correctly created.
			400 – Bad Request	Shows error message.

		that the new patient is correctly stored.	500 – Internal Server Error	-

This GET URI allows the user to look at the complete list of patients that are in the database. If the code returned after the request is “200”, it means the request went through correctly. However, if the code is different, it indicates that something went wrong. Code “404” indicates that the resource trying to be reached by the URI has not been found. Code “400” belongs to a bad request, whilst “500” indicates the error is in the system. The POST request is sent when the template fields are completed. If it they are filled out correctly, it will have a “201” created code and the new patient information will be stored in the database. If the template is filled out wrongly or has missing values, the code will be “400” and an error message will be displayed letting the user know that the form was filled out incorrectly.

4.2.2.3.2 Patient

The tables below describe the URIs that allow for the manipulation and management of a single patient, providing their ID. These include the creation, deletion, viewing and editing of the patient information.

shows how all the URIs for the resource *Patients* and *Patient* are interconnected to understand more easily what each one of them does.

Table 3 HTTP methods for the URI /patients/template

URI: /patients/template				
HTTP method	Request Body	Function	Response Code	Response Body
GET	-	Generate template fields.	200 – OK	Shows template with fields to fill

		Depending on the type, it will display a template for the creation of a patient, or for the search of a patient.		out.
			404 – Not Found	-
			400 – Bad Request	-
			500 – Internal Server Error	-

This URI is parametrised. What this means is that there is a parameter that is added at the end of it that will indicate what type of template to display. There are two possible types: `/patients/template?type=new,` which will show the template with fields to create a new patient, and `/patients/template?type=search` which will show a search field for the display of a patient given its id. The GET request renders the template which the user will fill out if the code is "200".

Table 4 HTTP methods for the URI /patients/:userID

URI: /patients/:userID				
HTTP method	Request Body	Function	Response Code	Response Body
GET	The ID of the patient to be shown.	Request that shows the representation of the patient with the corresponding ID.	200 – OK	Shows patient information with the corresponding ID.
			404 – Not Found	-
			400 – Bad	Shows error

			Request	message.
			500 – Internal Server Error	-
DELETE	ID of the patient that has to be deleted.	Request that sends the ID of the user to delete from the database.	200 – OK	Shows message stating the user ID if it was deleted.
			404 – Not Found	-
			400 – Bad Request	-
			500 – Internal Server Error	-

The GET request oversees the displaying the representation of patient given its ID. Through the DELETE request the patient information is deleted from the database given the userID. The ID is given by the template shown by previously explained URI `/patients/template?type=search`.

Table 5 HTTP methods for the URI /patients/template/:userID

URI: /patients/template/:userID				
HTTP method	Request Body	Function	Response Code	Response Body
GET	The userID.	Request that	200 – OK	Shows form fields

		shows the template fields corresponding to the user ID so that they can be changed.		with the patient information corresponding to the ID so that they can be edited.
			404 – Not Found	-
			400 – Bad Request	
			500 – Internal Server Error	-

This URI has only a GET request that oversees the displaying of a template that has the information of a patient given its ID. The ID is given by the user when it uses the previously explained URI The ID is given by the user when it uses the previously explained URI `/patients/template?type=search`. We were not able to use the UPDATE request, the why is explained in section 5.2.1.1 . This method uses the *create* method. It overwrites the patient with the ID and the information input, without allowing the user to change the ID.

4.2.2.3.3 All Raw Data

The table below explains the requests that manage all the Raw Data in the database, independent of which session it belongs to.

Table 6 HTTP methods for the URI /data

URI: /data

HTTP method	Request Body	Function	Response Code	Response Body
GET	-	Lists all the Raw Data in the database.	200 – OK	Shows list of Raw Data.
			404 – Not Found	-
			400 – Bad Request	Shows error message stating the Raw Data wasn't found
			500 – Internal Server Error	-

4.2.2.3.4 Data

The tables below show the requests needed to access groups of the Raw Data based on the session they belong to.

Table 7 HTTP methods for the URI /data/template

URI: /data/template				
HTTP method	Request Body	Function	Response Code	Response Body
GET	-	Generate HTML template for the introduction of	200 – OK	Renders small search bar to introduce the ID.
			404 – Not	-

		the session ID for which you want to see the data.	Found	
			400 – Bad Request	-
			500 – Internal Server Error	-

This request allows the user to introduce the session ID for which they want to see the raw data. If the code returned is “200”, then a search bar will be displayed, if not then there was an error either with the request or with the system.

Table 8 HTTP methods for the URI /data/:sessID

URI: /data/:sessID				
HTTP method	Request Body	Function	Response Code	Response Body
GET	The sessID.	Request that shows all the raw data of a session given its ID.	200 – OK	Displays all raw data of a session.
			404 – Not Found	-
			400 – Bad Request	Shows error message stating the session was not found.
			500 – Internal Server Error	-

This request shows all the raw data of a given session. If the code returned is "200", it will display the raw data of the session for which the ID was given by the user using the URI `/data/template` explained before. If the code returned is "400" it means it was a bad request, which means that no raw data was found for the session given, which means the session with the ID given does not exist in the database.

4.2.2.3.5 Sessions

The tables below shows all the HTTP requests that work with a group of sessions.

Table 9 HTTP methods for the URI /patients/sessions/data

URI: /patients/sessions/data				
HTTP method	Request Body	Function	Response Code	Response Body
GET	-	Generate template that allows the insertion of the document path and the user ID.	200 – OK	Shows template to fill out to create a new session.
			404 – Not Found	-
			400 – Bad Request	-
			500 – Internal Server Error	-
POST	Document path and userID.	Sends the information from the filled-	201 – OK	Shows message stating it was correctly created.

		out template to the framework to calculate the values of the session.	400 – Bad Request	Shows error message.
			500 – Internal Server Error	-

The GET request of the URI will show the template where the user will need to put the document address containing the Raw Data of related to that session and to a user that will also need to be input. The POST request sends the information to the method that will take the data in the file and calculate the session values. The algorithm used is explained in the *Algorithm creation* section. It will also then send the information calculated to the database to be stored.

Table 10 HTTP methods for the URI /patients/sessions/template

URI: /patients/sessions/template				
HTTP method	Request Body	Function	Response Code	Response Body
GET	-	Displays template to look for the sessions of a certain patient.	200 – OK	Shows search bar to introduce patient ID.
			404 – Not Found	-
			400 – Bad Request	-
			500 – Internal Server Error	-

This request provides a template so that the user can look for all the sessions of a given patient. If the code returned is “200”, then it will display a search bar to introduce the ID.

Table 11 HTTP methods for the URI /patients/:userID/sessions

URI: /patients/:userID/sessions				
HTTP method	Request Body	Function	Response Code	Response Body
GET	The userID.	Lists all the Sessions of a certain patient.	200 – OK	Shows list of Sessions.
			404 – Not Found	-
			400 – Bad Request	Shows error message stating the Sessions for that patient weren’t found.
			500 – Internal Server Error	-

This is a request that is used to manage the sessions of a certain patient given their ID in the form using the URI `/patients/sessions/template` previously explained. If the code is “200”, then the list of sessions corresponding to the patient ID will be displayed, if the code returned is “400”, then the request was wrong, meaning that there are no sessions in the database related to the patient with the ID introduced.

The URI is a representation of the relationship between resources; therefore, it also includes the resource patients as the session belongs to a certain patient.

4.2.2.3.6 Session

The table below explains all the URIs that manage a single session given its ID, and the patient's ID.

Table 12 HTTP methods for the URI /patients/:id/sessions/:idS

URI: /patients/:userID/sessions/:sessID				
HTTP method	Request Body	Function	Response Code	Response Body
GET	The patient ID and the session ID.	Lists all the Sessions of a certain patient.	200 – OK	Shows session info corresponding to the patient and session ID.
			404 – Not Found	-
			400 – Bad Request	Shows error message stating the Session for that patient wasn't found.
			500 – Internal Server Error	-

This is a request that also follows the hierarchical syntax explained before. It manages a single session at a time given its ID, and the patient's ID. This URI is related

to the expert system, and its full functionality will be later explained in the next section as well as in the Implementation section.

4.2.2.4 How the URIs work together

The following figures show how the URIs of each resource work together, both in the frontend part of the application (the templates) and the backend part.

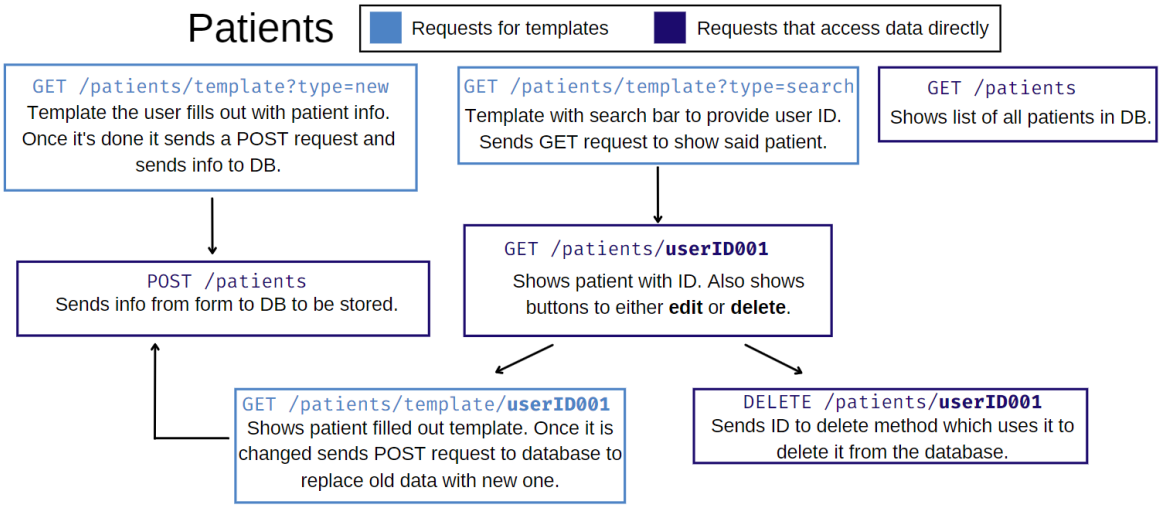


Figure 24 URIs for the resources *Patients* and *Patient* working together.

Figure 24 shows all the URIs that refer to a patient or set of patients. In light blue, those URIs that provide a template where data must be input are shown. The set of requests can start with creating a new patient, then sending a POST request; searching for a patient, then either editing or deleting that patient with a POST or DELETE request; or finally with a GET request that shows a patient.

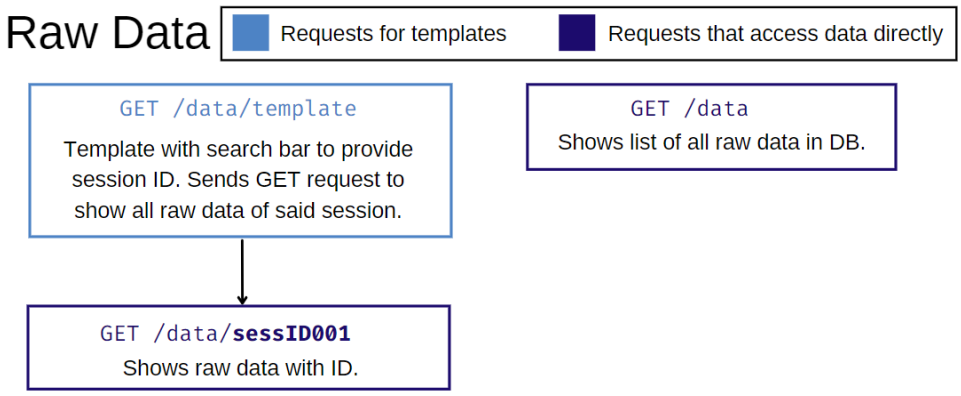


Figure 25 URIs for the resources *All Raw Data* and *Data* working together.

Figure 25 shows the three requests related to the Raw Data resource. Only one is used for the frontend part, which allows the user to introduce an ID to then request a GET for a particular session. The other GET request shows a set of Raw Data.

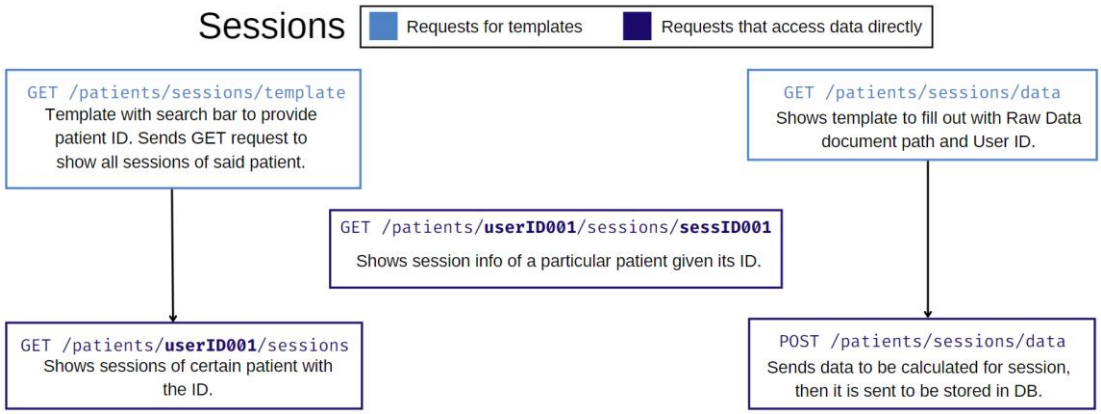


Figure 26 URIs for the resources *Sessions* and *Session* working together.

Figure 26 shows the how the URIs referring to a Session or set of Sessions work. To either show sessions of a patient, or create session data, a template must first be filled out in the frontend part of the application. There is another GET request that accesses a Session of a certain Patient.

4.2.2.5 Algorithm creation

This algorithm is in charge of calculating useful values from thousands of EMG recordings. These values are not only more useful for the therapist as they are representative of many EMG recordings, but they are also needed for the expert system. An expert system cannot be fed thousands of values to reach a conclusion. The knowledge base is composed of a few but important parameters that represent the data and can be used to reach valuable conclusions.

The EMG values are used to calculate other values that will be used to achieve the muscle activity. First, the EMG signals must be rectified to obtain the absolute value. After rectifying them the first calculation is the RMS using the formula shown in Figure 27. *Wsize* refers to the size of the moving window (number of samples in the window) for which the RMS will be computed. The maximum of these signals will result in the MVC values.

$$x_{\mathrm{rms}} = \sqrt{\frac{{x_1}^2 + {x_2}^2 + \cdots + {x_n}^2}{n}}$$

Figure 27 RMS formula

Finally, the muscle activity is calculated by computing the area below the curve for each muscle. Then divide each of the areas by the sum of them all, multiplied by 100 to obtain the percentages.

4.2.3 Microservice 1: Expert System

We will firstly explain the Expert System proposed, then how the API of the RESTful Web Service was designed.

4.2.3.1 The Expert System

The expert system is in charge of providing the rules that will decide the correct course of treatment. All rules have similar structures, what changes between

them is the input data and how it is considered, therefore changing the conclusion drawn.

Since what we are trying to determine is how or if the treatment should change given certain input values, there will be as many rules as to cover each possible input. As it has been previously explained, the rule-based system will take into consideration the diagnosis (whether the patient has biceps or triceps atrophy), if the exoskeleton was set to assist or resist biceps flexion, the number of elastics used in that session, and the muscular activity that was obtained in the session with those parameters. As a result, the expert system will compare the values obtained with what they should be given the parameters input and will determine if the treatment (number of elastics used) should be changed or maintained.

In section 3.1.4 we established that the muscle activity should be a parameter to predict future sessions, along with the exoskeleton configuration (the number of elastics used and whether they are positioned to assist or resist biceps). However, this parameter alone would not provide sufficient information, so it was decided to add another parameter we named “diagnosis”. This parameter describes what the patient suffers from. It will determine the muscle which the session will focus on and will also influence which values of muscular activity are used for the prediction of future therapies. All these parameters determine how many rules there will be. As shown in Figure 28, there are twelve possible conclusions, and therefore there should be twelve rules, one per result. The conclusions are drawn based on previously calculate values of what the muscular activity should be.

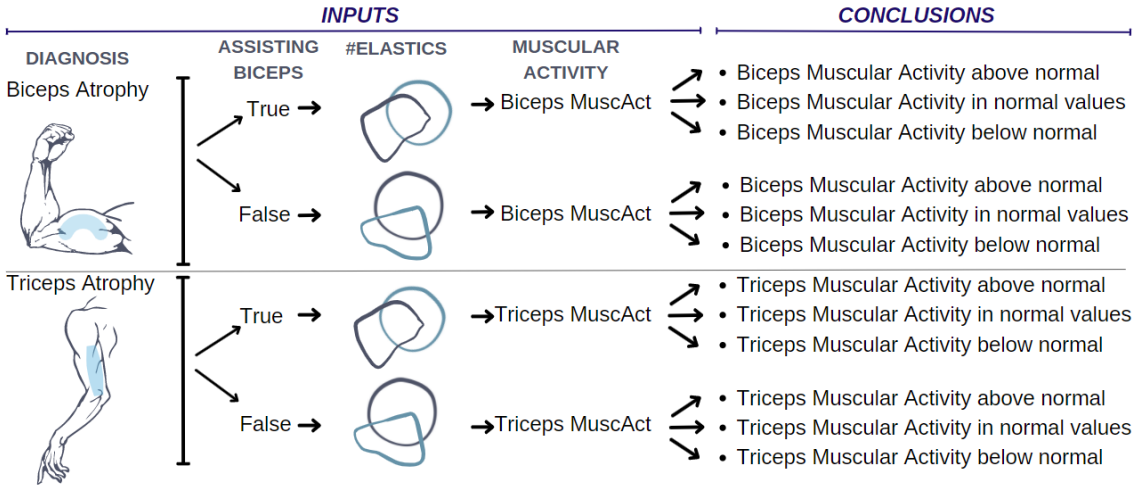


Figure 28 All possible inputs and conclusions in the Rule-Based system.

Considering we are using a rule-based system, an approximation of the rules we will use will be as follows:

```
IF diagnosis=Biceps Atrophy and Assisting Biceps=True and
#Elastics=X and Muscular Activity=Y THEN Biceps Muscular
Activity is too low (number of elastics should be higher).
```

4.2.3.2 Identification and RESTful API design of Resources

All resources used in this Web Service have been previously explained. This microservice will be using the Session and Sessions resources, and by default it will therefore use the Patient resource due to the hierarchical nature of the URIs. This Web Service has a single URI; however, it does use some of MS2 URIs to get the data it needs, this will be explained further in the Implementation section of the paper.

- **/sess/:userID/:sessID/:elastics/:diag/:assist:** it accesses the session information and calculated predictions using the User ID, Session ID, and the parameters (number of elastics, diagnosis, and biceps assistance).

Table 13 HTTP methods for the URI /sess/:userID/:sessID/:elastics/:diag/:assist:

URI: /sess/:userID/:sessID/:elastics/:diag/:assist:				
HTTP method	Request Body	Function	Response Code	Response Body
GET	-	Lists all the info of a session along with the calculations and predictions.	200 – OK	Shows session info corresponding to the patient and session ID, and the calculations given the parameters.
			404 – Not Found	-
			400 – Bad Request	Shows error message stating the Session for that patient wasn't found.
			500 – Internal Server Error	-

This URI only has a GET request. The request body is used to get the information of the session by connecting to MS2 and sending the following request: `GET /patients/:userID /sessions/:sessID.` This way MS1 can access the data stored inside each session so that it can be utilised by the expert system. The rest of the request body are the parameters that are also used by the rule-based system to reach a conclusion.

4.2.3.3 How the Microservices work together

Figure 29 shows how both microservices work together when the example request `/sess/user001/sess001/5/TA/true` is sent.

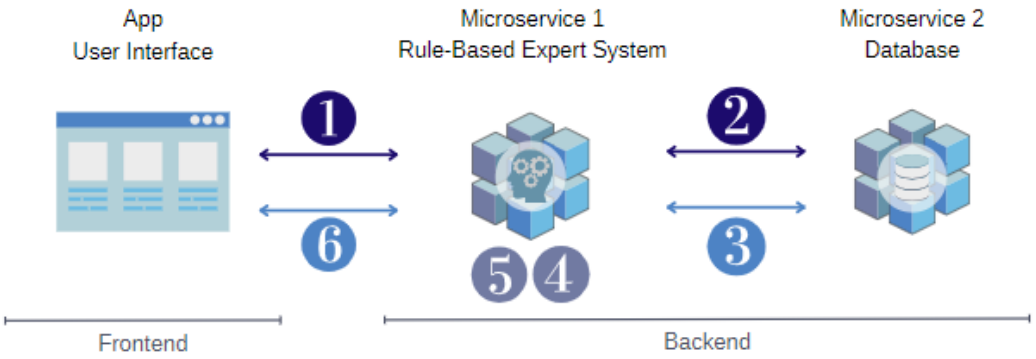


Figure 29 How the microservices work together

1. Through the User Interface, the user inserts the information: userID user001, sessionID sess001, 5 elastics, a diagnosis of Triceps Atrophy, and assisting biceps. This request is received by MS1.
2. MS1 sends this GET request: `/patients/user001/sessions/sess001` to MS2.
3. MS2 returns the data from the session with id sess001.
4. The data obtained from the database is asserted into the Working Memory of the Expert System.
5. Conclusions are drawn based on the facts provided.
6. These conclusions are displayed int the User Interface.

4.3 Frontend Design

The User Interface will consist of various pages with different functionality. It will look like a mainstream website, with a navigation bar where the user will be able to choose which section of the application they would like to use. The figures below show the wireframe for each section, which are Sessions, Patients, and Raw Data, corresponding to each resource.

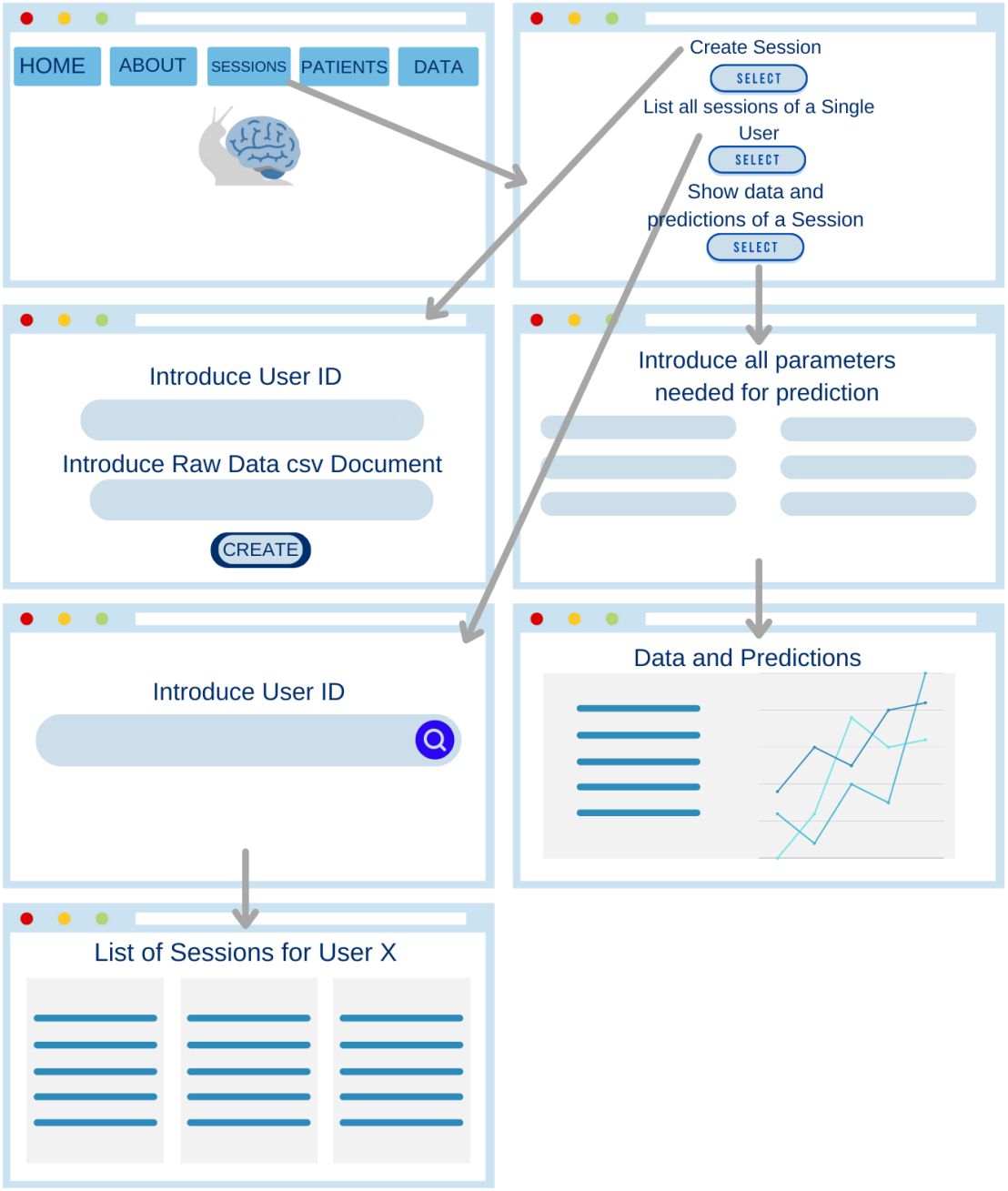


Figure 30 Wireframe for the *Sessions* resource part of the application.

Figure 30 shows all the frontend parts that apply to the *Session* and *Sessions* resources. It will first display the operations available for this resource (create, list sessions, show predictions). Each of these will display a template that will be filled out, and once it is done it will carry out the operation.

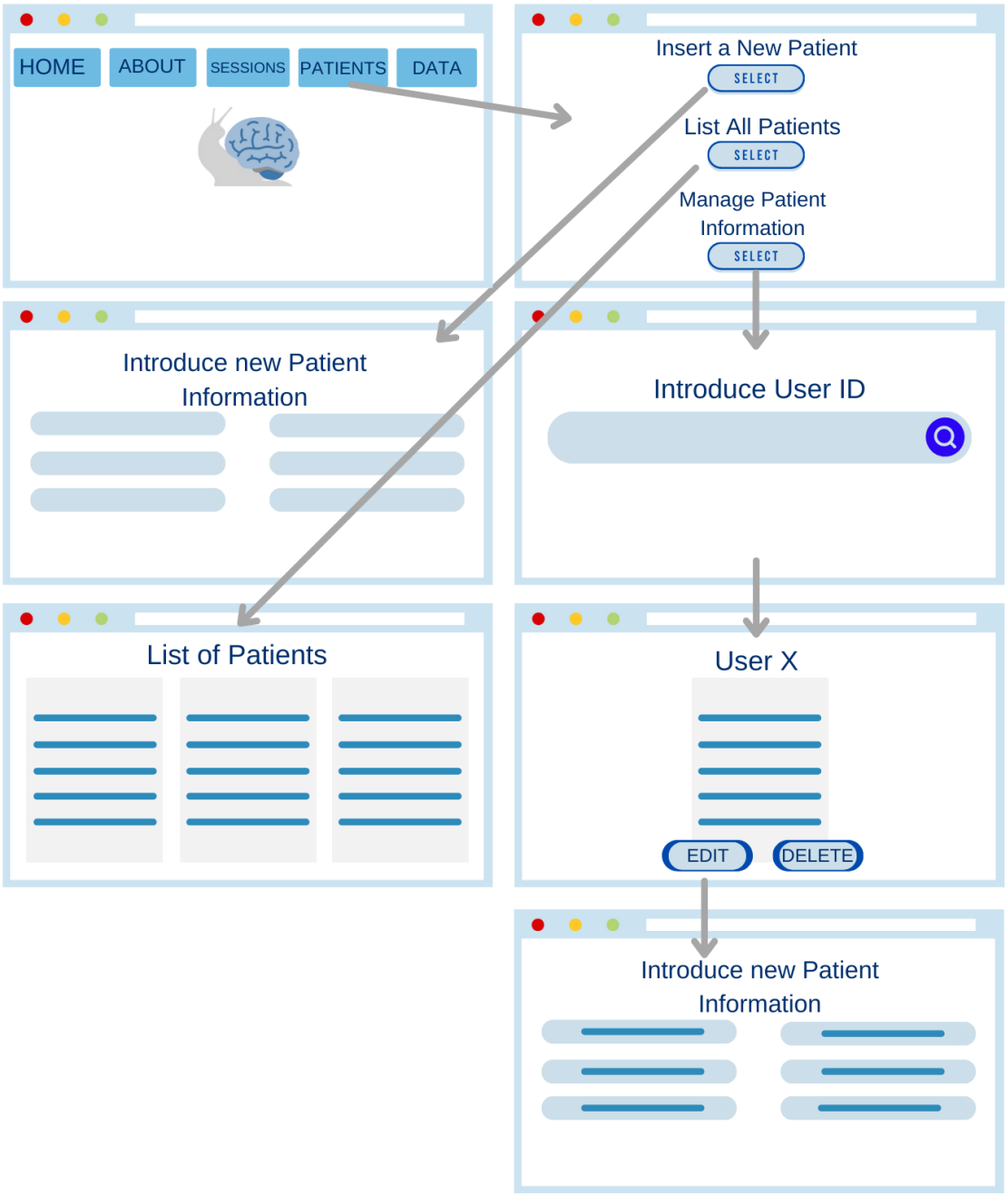


Figure 31 Wireframe for the *Patients* resource part of the application.

Figure 31 shows the wireframe applied to the resources *Patient* and *Patients*. Similarly, to the resources *Session* and *Sessions*, the operations related to these resources are displayed. Once one is chosen, a template will be shown which will be filled out with the information needed to carry out the operation.

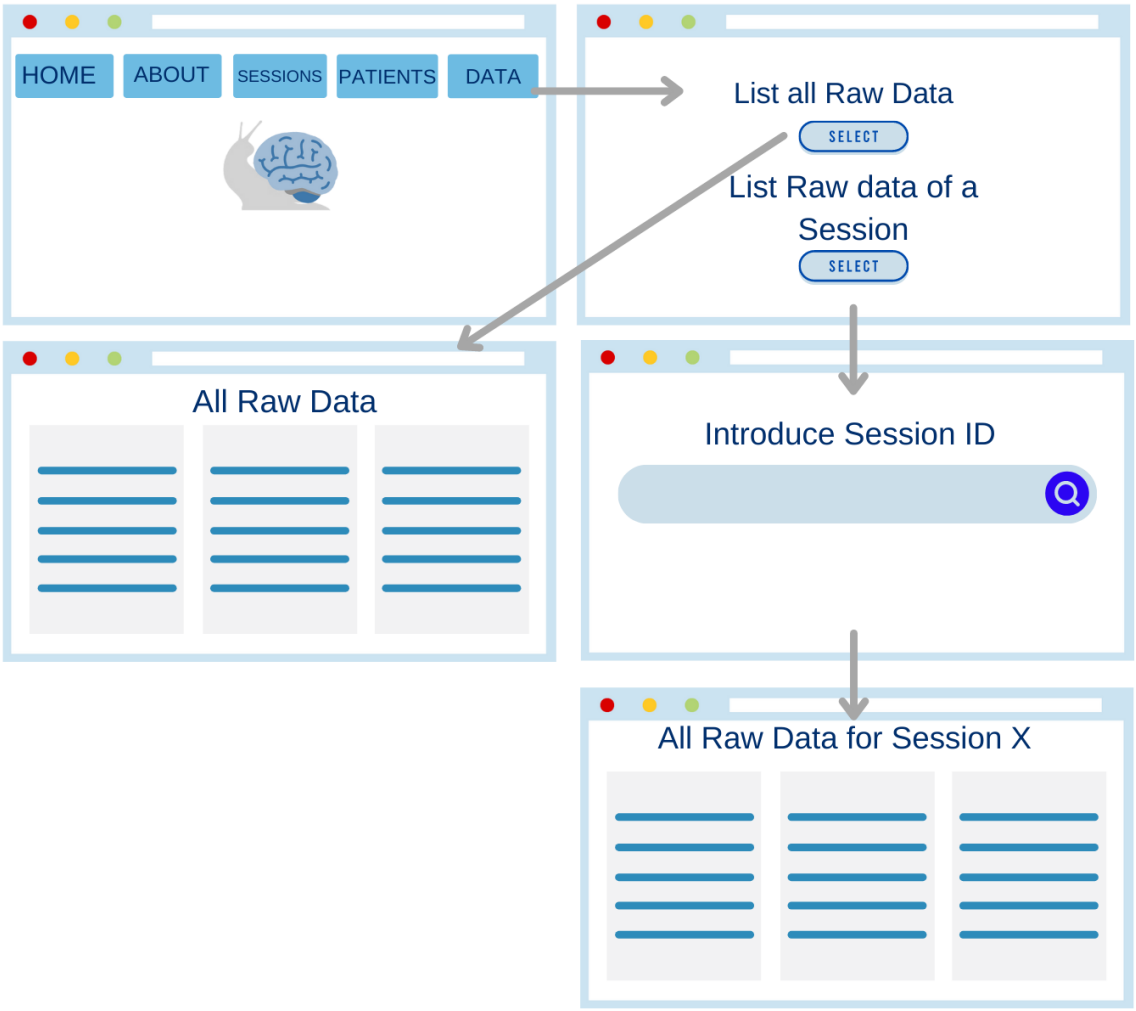


Figure 32 Wireframe for the *Data* resource part of the application.

Figure 32 shows the wireframe that applies to the resources *Data* and *All Raw Data*. It will work in the same way as the previous two, carrying out the operations once the data that is needed is inserted in the templates.

5 IMPLEMENTATION

This section will explain everything that is needed to carry out the implementation of the application. The technologies used will be explained separately for each microservice. Although both use the same framework, other technologies are used and implemented with it, and will therefore be more intuitive to see how they all work together in each microservice.

5.1 Backend Implementation

We will begin explaining the inner workings of the application. To maintain the format of the paper, we will start by explaining the backend implementation of MS2, the Web Service in charge of accessing the database. The code for the implementation of this microservice can be found at https://github.com/sofiagrandia/cassandra-upperlimb.

5.1.1 Microservice 2: Database

In this section we will explain the main components of Play framework, and how they are structured. These components also include other technologies that are used for accessing and processing the data in this microservice, such as R (using the ScriptEngine interface with Renjin's R interpreter for java) and Cassandra (using DataStax's driver and interfaces).

5.1.1.1 Database Implementation

Cassandra is the most popular wide-based non-relational database. It is highly open source and highly tolerant to failures. Although Cassandra is schema-less, it allows the storage of data in a similar way to relational databases, and although it does not offer join operations, it can be designed around these limitations [16].

To understand better how it is implemented, Cassandra's data model is shown in Figure 33. Each table represents a resource, and these are grouped inside a *keyspace* we named *upperlimb*. These keyspaces are then grouped into clusters that can span various nodes. The tables also have *primary keys*, which work in the same way as

relational databases. They may also have *clustering keys*, which control how data is sorted for storage, allowing for a faster retrieval of data.

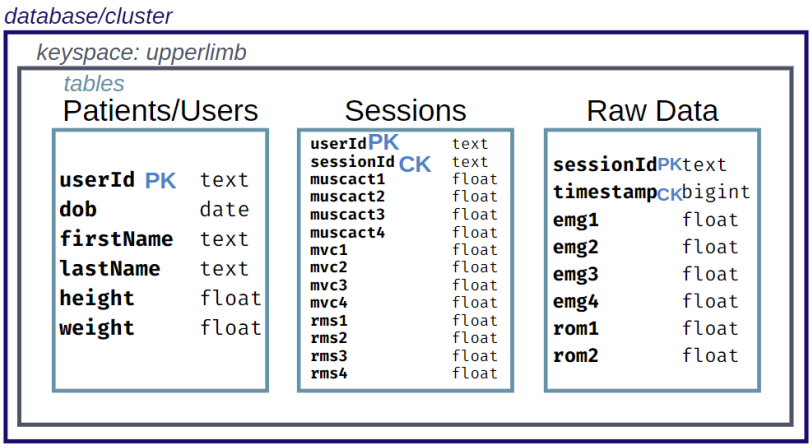


Figure 33 Cassandra's data model applied to our data.

5.1.1.2 RESTful Web Service Implementation

Play Framework is an open-source framework for Java and Scala web applications. It provides all the necessary components for the creation of a RESTful Web Service, which is why it is one of the most popular frameworks for the creation of these apps. They also include SBT support, which is a build tool for Scala and Java projects. It provides the use of a *build.sbt* file which allows the compilation of the project with all the necessary libraries and files.

5.1.1.2.1 Router

Play framework will initially analyse the URI it receives to check if it can process the request. This is done by comparing the URI to those found in the document */routes*, found in the */conf* directory. The URIs and API for this microservice were previously explained in *RESTful API resource design*.

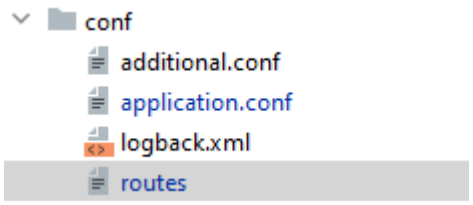


Figure 34 File structure in the Play framework configuration

This file contains the three components that are used to process the requests that were defined in section 4.2.2: the method of the request, the URI for said method, and the controller method it calls.

```
GET     /patients                       controllers.DBController.listPatients()
POST    /patients                       controllers.DBController.createPatient(request:Request)

DELETE  /patients/:userID               controllers.DBController.deletePatient(userID:String)
GET     /patients/template              controllers.DBController.template(type: String)
GET     /patients/:userID               controllers.DBController.retrievePatient(userID:String)
GET     /patients/template/:userId      controllers.DBController.editPatient(userId:String,request:Request)

GET     /data/template                  controllers.DBController.searchData()
GET     /data                           controllers.DBController.listAllRawData()
GET     /data/:sessID                   controllers.DBController.listRawDataBasedOnSess(sessID:String)

GET     /patients/sessions/template             controllers.DBController.searchSess()
GET     /patients/:userID/sessions              controllers.DBController.listSessions(userID:String)
GET     /patients/:userID/sessions/:sessID      controllers.DBController.retrieveSession(userID:String,sessID:String)
GET     /patients/sessions/data                 controllers.DBController.createGetSess()
POST    /patients/sessions/data                 controllers.DBController.createSession(request:Request)
```

Figure 35 Routes file in Play framework

5.1.1.2.2 Controller

The controller has all the code that is executed when a request is received. It is in charge of managing requests, returning the representation of the response defined in API (explained in section 4.2.2.3) The figures below show the methods that are in charge of executing for the URI http://localhost:9000/patients/template?type=new, and how the database is accessed. It also shows how the HTML forms, explained further in the Frontend implementation section, work.

```
public Result createGet(){
    Form<Patient> patientForm = formFactory.form(Patient.class);

    return ok(newPatient.render(patientForm));
}
```

Figure 36 Controller method called for the URI GET /patients/template?type=new

Figure 36 shows the method used to show the form that will be filled out by the therapist with the user information, whilst Figure 37 shows how the information from that form is retrieved and stored in the database using CQL (Cassandra's Query Language) with DataStax's interfaces.

```
public Result createPatient(Http.Request request) {

    DynamicForm form = formFactory.form().bindFromRequest(request);
    if (form.hasErrors()) {
        System.out.println("Please Correct the Form Below");
        System.out.println(form.errorsAsJson());
        logger.error("errors = {}", form.errors());
        return badRequest(messageCancel.render("Check the form for errors/missing values. Click accept to close window. Click cancel to edit."));
    }

    CqlSession session = CqlSession.builder()
            .build();
    Patient patient = new Patient();
    try {
        patient.setUserID(form.get("userID"));
        patient.setFirstName(form.get("firstName"));
        patient.setLastName(form.get("lastName"));
        patient.setWeight(Float.parseFloat(form.get("weight")));
        patient.setHeight(Float.parseFloat(form.get("height")));
        patient.setDob(LocalDate.parse(form.get("dob")));
    }catch(NumberFormatException e){
        return badRequest(messageCancel.render("Check the form for errors/missing values. Click accept to close window. Click cancel to edit."));
    }

    LocalDate date = patient.getDob();
    StringBuilder sb = new StringBuilder("INSERT INTO upperlimb.user (userID, firstName, lastName, height, weight, dob) VALUES ('")
            .append(patient.getUserID()).append("', '").append(patient.getFirstName()).append("', '")
            .append(patient.getLastName()).append("', ").append(patient.getHeight()).append(", ")
            .append(patient.getWeight()).append(", '").append(date.getYear()).append("-").append(date.getMonthValue())
            .append("-").append(date.getDayOfMonth()).append("');");

    String query = sb.toString();
    session.execute(query);
    session.close();
    return ok(message.render("Patient created/updated correctly"));

}
```

Figure 37 Controller method called for the URI POST /patients

CqlSession is the main entry point for the Cassandra DataStax driver, and it is what will be used to execute Cassandra queries. The DataStax driver defaults to `127.0.0.1:9042`, which is the Native Transport Port that is defined in the Cassandra *.yaml* file and is charge of listening to CQL clients. The name of the keyspace must also always be included when referencing a table i.e. `INSERT INTO upperlimb.user`.

After checking if the form was correctly filled out, this method takes the information that was passed by the form and stores it in the Patient object. Then the information stored in *patient* is used to build the string that will be used as a query for the database. This string specifies the keyspace *upperlimb*, and table of the Cassandra database plus all the information needed to complete the query. This string is then sent to Cassandra using *session.execute*.

There is a method in the controller corresponding to the URI POST /patients/sessions/data, that uses an R method to calculate the session's values. This method is called from the method *createSession* shown in Figure 38. This method, *sessionR*, is shown in Figure 39.

```
public Result createSession(Http.Request request) throws Exception {
    DynamicForm form = formFactory.form().bindFromRequest(request);

    if (form.hasErrors()) {
        System.out.println("danger Please Correct the Form Below");
        System.out.println(form.errorsAsJson());
        logger.error("errors = {}", form.errors());
        return badRequest(messageCancel.render("Check the form for errors/missing values. " +
                "Click accept to close window. Click cancel to edit."));
    }
    String data;
    String id;
    try {
        data = (form.get("doc"));
        id = (form.get("userID"));

    }catch(NumberFormatException e){
        return badRequest(messageCancel.render("Check the form for errors/missing values. " +
                "Click accept to close window. Click cancel to edit."));
    }
    sessionR(data);
    System.out.println(data);
    System.out.println(id);

    CqlSession session = CqlSession.builder()
            .build();

    DateFormat dformat = new SimpleDateFormat( pattern: "YYYY-MM-dd HH:mm:ss");
    Date date = new Date();
    ObjectMapper om = new ObjectMapper();
    System.out.println(rms1);
    session.execute("INSERT INTO upperlimb.sessions (userid, sessionid, rms1, rms2, rms3, rms4, " +
            "mvc1, mvc2, mvc3, mvc4, muscact1, muscact2, muscact3, muscact4) " +
            "VALUES('" +id+"','"+sessionId+"',"+rms1+","+rms2+","+rms3+","+rms4+","+mvc1+","+mvc2+","+
            mvc3+","+ mvc4+","+muscact1+","+muscact2+","+muscact3+","+muscact4+");");
    session.close();

    return ok(message.render("Session with id: " + sessionId + " created"));
}
```

Figure 38 Method called for the URI POST /patients/sessions/data

The values inserted into the database are calculated in *sessionR*, shown in Figure 39 in red. The interface *ScriptEngine* allows the use of the *Renjin* interpreter. This allows to use the R language from a Java application. Using *engine.eval*, the R language can be used directly from this method, and the objects returned can be parsed into any type we need, in our case we parse them into *Float*. The method begins by importing the CSV file that was passed as an argument and reading it into a Data Frame using the sampling rate of 1000Hz. It uses the library *biosignalEMG* that allows the manipulation of EMG signals.

```
public Result sessionR(String data) throws Exception{

    ScriptEngineManager factory = new ScriptEngineManager();
    ScriptEngine engine = factory.getEngineByName("Renjin");

    engine.eval("library('biosignalEMG')");
    engine.eval("prueba2 <- read.csv(\"C:/"+data+"\", sep = \";\")");
    engine.eval("emgDF<-data.frame(prueba2$EMG1, prueba2$EMG2, prueba2$EMG3, prueba2$EMG4)");
    Object obj = engine.eval("sessionid=as.vector(prueba2$sessionID[1])");
    engine.eval("pruebaemg<-as.emg(emgDF, samplingrate=1000)");

    engine.eval("pruebaemg_rectif1=rectification(pruebaemg, channel=1)");
    engine.eval("pruebaemg_rectif2=rectification(pruebaemg, channel=2)");
    engine.eval("pruebaemg_rectif3=rectification(pruebaemg, channel=3)");
    engine.eval("pruebaemg_rectif4=rectification(pruebaemg, channel=4)");
    engine.eval("pruebaemgRMS1=envelope(pruebaemg_rectif1, method=c(\"RMS\"), wsize=50)");
    engine.eval("pruebaemgRMS2=envelope(pruebaemg_rectif2, method=c(\"RMS\"), wsize=50)");
    engine.eval("pruebaemgRMS3=envelope(pruebaemg_rectif3, method=c(\"RMS\"), wsize=50)");
    engine.eval("pruebaemgRMS4=envelope(pruebaemg_rectif4, method=c(\"RMS\"), wsize=50)");
    engine.eval("pruebaRMSna1 <- na.omit(pruebaemgRMS1)");

    Object obj1 = engine.eval("RMS1 = mean(na.omit(unlist(pruebaemgRMS1[1])))");
    Object obj2 = engine.eval("RMS2 = mean(na.omit(unlist(pruebaemgRMS2[1])))");
    Object obj3 = engine.eval("RMS3 = mean(na.omit(unlist(pruebaemgRMS3[1])))");
    Object obj4 = engine.eval("RMS4 = mean(na.omit(unlist(pruebaemgRMS4[1])))");

    rms1 = Float.parseFloat(obj1.toString());
    rms2 = Float.parseFloat(obj2.toString());
    rms3 = Float.parseFloat(obj3.toString());
    rms4 = Float.parseFloat(obj4.toString());

    Object obj5 = engine.eval("MVC1 = max(na.omit(unlist(pruebaemgRMS1[1])))");
    Object obj6 = engine.eval("MVC2 = max(na.omit(unlist(pruebaemgRMS2[1])))");
    Object obj7 = engine.eval("MVC3 = max(na.omit(unlist(pruebaemgRMS3[1])))");
    Object obj8 = engine.eval("MVC4 = max(na.omit(unlist(pruebaemgRMS4[1])))");

    mvc1 = Float.parseFloat(obj5.toString());
    mvc2 = Float.parseFloat(obj6.toString());
    mvc3 = Float.parseFloat(obj7.toString());
    mvc4 = Float.parseFloat(obj8.toString());

    engine.eval("pruebaemgI1=integration(pruebaemg_rectif1)");
    engine.eval("pruebaemgI1_final=unlist(pruebaemgI1[1])");
    engine.eval("A1=pruebaemgI1_final[length(pruebaemgI1_final)]");
    engine.eval("pruebaemgI2=integration(pruebaemg_rectif2)");
    engine.eval("pruebaemgI2_final=unlist(pruebaemgI2[1])");
    engine.eval("A2=pruebaemgI2_final[length(pruebaemgI2_final)]");
    engine.eval("pruebaemgI3=integration(pruebaemg_rectif3)");
    engine.eval("pruebaemgI3_final=unlist(pruebaemgI3[1])");
    engine.eval("A3=pruebaemgI3_final[length(pruebaemgI3_final)]");
    engine.eval("pruebaemgI4=integration(pruebaemg_rectif4)");
    engine.eval("pruebaemgI4_final=unlist(pruebaemgI4[1])");
    engine.eval("A4=pruebaemgI4_final[length(pruebaemgI4_final)]");
    engine.eval("AreaTotal=A1+A2+A3+A4");

    Object obj9 = engine.eval("muscact1=(A1/AreaTotal)*100");
    Object obj10 = engine.eval("muscact2=(A2/AreaTotal)*100");
    Object obj11 = engine.eval("muscact3=(A3/AreaTotal)*100");
    Object obj12 = engine.eval("muscact4=(A4/AreaTotal)*100");

    muscact1 = Float.parseFloat(obj9.toString());
    muscact2 = Float.parseFloat(obj10.toString());
    muscact3 = Float.parseFloat(obj11.toString());
    muscact4 = Float.parseFloat(obj12.toString());

return ok(ApplicationUtil.createResponse("R", true)); }
```

Figure 39 Method *sessionR* where the session values are calculated using R.

The values imported into the data frame are then manipulated to achieve the values we want, and how each of these values are calculated was previously explained in the *Algorithm creation* section.

5.1.1.2.3 Request Management and Responses

Depending on the request, different responses are generated. As mentioned before, some requests are only used to generate forms for the user to input data into. These requests will look like the one shown in Figure 40.

```
public Result searchData(){
    DynamicForm patientForm = formFactory.form();

    return ok(retrieveRawData.render());
}
```

Figure 40 Method that returns a form used by the URI GET /data/search

Other methods, like the one shown in Figure 38, will return a message. If the request went through correctly, the method would return *ok* accompanied with the message that it was correctly done (code 200 or 201 as explained in previous sections). However, if there was an error, it will return a *badRequest* along with an error message (code 400).

5.1.2 Microservice 1: Expert System

As in section 5.1.1, the main components of Play framework will be explained, along with other technologies used such as Drools (rule-based system) and JavaWS (implementation used to connect with MS2). This microservice also provides the main components of the frontend part of the application and will therefore have two controllers: one that manages the expert system (DroolsController), and another in charge of the frontend (FrontendController, explained in section 5.2.2.1). The code for the implementation of this microservice can be found at https://github.com/sofiagrandia/drools-upperlimb.

5.1.2.1 Expert System Implementation

Drools is an open-source rule-based engine which the pattern-matching algorithm *Rete*. . It uses its own language called DRL. Its rules are stored in a *.drl* file, which will be used to generate our knowledge base.

Firstly, we create a new session inside a container where the rules (knowledge base) will be found. Then we set the values we and insert them into the rules. We fire all the rules and get the recommended treatment as a result.

There is a class *DiagInfo* () that is inserted into the rules. This class contains all the parameters needed for the prediction in a single class.

```
public class DiagInfo {

    private String diag;
    private float muscComp1;
    private float muscComp2;
    private Boolean assist;
```

Figure 41 Class DiagInfo

To be able to get the recommendation outside Drools, we created an object called *Rec* that can be inserted into the session and changed, then the information can be accessed from the Java method once it has the actual recommendation. This class has two attributes: a string called rec, where the recommendation will be stored, and status which will be changed once the recommendation has been changed. The initial string passed is the default one “start”, which will always be changed once the rules are fired.

```
public class Rec {

    public static final int INIT  = 0;
    public static final int DONE = 1;

    private String rec;
    private int status;
```

Figure 42 Class Rec

These classes are inserted and used by the rules in the file *rules.drl ()*. Each rule has a title name. The rules check the parameters that are input in the *when* statement, and if they are all correct then that rule is fired using the *then* statement. The recommendation string is modified with the correct procedure, and the status is changed

to *DONE* so that no more rules are triggered. This object can then be accessed to display the result in the user interface.

```
rule "Triceps Atrophy Assist High"
dialect "mvel"
  when
      r: Rec(status== Rec.INIT, rec: rec)
      s: Sess($muscAct: muscAct2)
      di : DiagInfo(diag=="TA",assist==true,$muscAct<muscComp1)
  then
      modify ( r ) { rec = "Muscular Activity of session:" +s.getMuscAct2+ ". Should be between values" +
      " [" + di.getMuscComp2 + " , " + di.getMuscComp1 +"]." + " Muscular activity too low, increase number of elastics",
      status=Rec.DONE};
      System.out.println("Muscular Activity of session:" +s.getMuscAct2);
      System.out.println("Should be between values" + " [" + di.getMuscComp2 + " , " + di.getMuscComp1 +"]");
      System.out.println("Muscular activity too low, increase number of elastics" );
end
```

Figure 43 Example of a Drools rule in the *rules.drl* file

5.1.2.2 RESTful Web Service Implementation

5.1.2.2.1 Router

As explained before, the */routes* file contains all the URIs that process the request. It is found in the */conf* directory, where the file with the rules (*/rules.drl*) is also found.

This microservice has only two URIs in its router, one for the expert system, and another for the frontend. They are shown in Figure 44.

```
# Homepage
GET     /                           controllers.FrontendController.index()

# Expert System
GET  /sess/:userId/:sessionId/:elastics/:diag/:assist  controllers.DroolsController.simpleGetSess(userId:String,sessionId:String,elastics:Float,diag:String,assist:Boolean)
```

Figure 44 Routes file for MS1

5.1.2.2.2 Controller

The *DroolsController* only has one method, this method oversees communicating with MS2 to obtain the data, calculate the margin values for the muscle activity, and communicate with the rule-based system. As it can be seen in Figure 39 the margins of the muscular activity are calculated according to the diagnosis and if the exoskeleton is assisting flexion. The calculations are surrounded by an orange box. As it has been explained before, the margins are set to be ±0.5 the function calculated in the previous section. These values, labelled as muscComp1 and muscComp2 are then used

by the expert system. They are passed through using the knowledge base session as shown in Figure 39 surrounded by a red box. This method also uses JavaWS to communicate with MS2. It is done by creating a class we named *MyClient*. The communication part of the method is shown in purple in Figure 39 and is further explained in the next section.

How the connection is established with MS2 and with Drools, along with the classes (showed in purple and red in Figure 45) used to support the connection will be explained more thoroughly in the following sections.

```
public Result simpleGetSess(String userId, String sessionId, Float elastics, String
diag, Boolean assist) throws Exception {

    Float muscComp1 = 0f;
    Float muscComp2 = 0f;
                                                        Connection with MS2
    MyClient client = new MyClient(ws);
    JsonNode jn = client.jsonNodeSess(userId, sessionId).toCompletableFuture().get();
    JsonNode in = jn.get("response");

    try{
        System.out.println(in.get("userID").asText());
    }catch (NullPointerException npe){
        return ok("SESSION NOT FOUND, check session or user ID");
    }

    Sess sess = Json.fromJson(in, Sess.class);
                                                        Calculate margins
    if(diag.equalsIgnoreCase("BA")){
        if(assist==true) {
            muscComp1 = (-2.5755f * (elastics-0.5f)) + 46.968f;
            muscComp2 = (-2.5755f * (elastics+0.5f)) + 46.968f;
        }else{
            muscComp1 = (1.0088f*(elastics-0.5f)) + 46.97f;
            muscComp2 = (1.0088f*(elastics+0.5f)) + 46.97f;}
        }else{
        if(assist==true) {
            muscComp1 = (3.5938f * (elastics-0.5f) + 12.846f);
            muscComp2 = (3.5938f * (elastics+0.5f) + 12.846f);
        }else{
            muscComp1 = (-0.1827f *(elastics-0.5f) + 12.74f);
            muscComp2 = (-0.1827f *(elastics+0.5f) + 12.74f);}
        }

    String prettyString = jn.toPrettyString();
    System.out.println(prettyString);
    String res = new String();
    DiagInfo di = new DiagInfo();                       Connection with Drools
    try {
        KieServices ks1 = KieServices.Factory.get();
        KieContainer kContainer1 = ks1.newKieClasspathContainer();
        KieSession kieSession1 = kContainer1.newKieSession("upperlimb-rules");

        final Rec rec= new Rec();
        final DiagInfo diagInfo = new DiagInfo();
        diagInfo.setDiag(diag);
        diagInfo.setMuscComp1(muscComp1);
        diagInfo.setMuscComp2(muscComp2);
        diagInfo.setAssist(assist);
        rec.setRec("hello, world");
        rec.setStatus(Rec.INIT);

        kieSession1.insert(sess);
        kieSession1.insert(diagInfo);
        kieSession1.insert(rec);
        kieSession1.fireAllRules();
        res = rec.getRec();
    } catch (Throwable t) {
        t.printStackTrace();
    }
    SessDisplay sd=new SessDisplay(sess, res, elastics, diag, assist);
    return ok(showResult.render(sd));
}
```

Figure 45 DroolsController Method

5.1.2.2.3 Connection with MS2

To connect to the other microservice, we use the *WSClient* interface that is included in Play framework. This allows to make HTTP asynchronous calls to other Web Services. We included this in a class we called *MyClient* (as shown in purple in Figure 45), that includes all the methods we need to communicate with MS2.

```
public class MyClient implements WSBodyReadables, WSBodyWritables{
    private final WSClient ws;

    @Inject
    public MyClient(WSClient ws) { this.ws = ws; }

    public CompletionStage<JsonNode> jsonNodeSess(String userId, String sessionId){
        String idUrl = "http://localhost:9000/patients/"+userId+"/sessions/" + sessionId;
        return ws.url(idUrl).get().thenApply(WSResponse::asJson);
    }
```

Figure 46 *MyClient* class that communicates with MS2

This class sends a request to MS2 that is running in port 9000, then receives the response as JSON, which is later transformed into a *Sess* object. This session object will be the one studied by the expert system.

5.1.2.2.4 Request Management and Responses

Similarly to MS2, the method shown in Figure 45 will display the result of the expert system using HTML, if the response code is 200 (Figure 58).

5.2 Frontend Implementation

Although the app has two microservices, in the frontend implementation they are joined seamlessly, only distinguished by the URIs as the ones for MS1 begin with http://localhost:8080/ (port 8080) whilst the ones for MS2 start with http://localhost:9000/ (port 9000). However, the HTML forms are found in different microservices depending on the resources used. The following sections explain how the user interface works in each microservice although they mostly share the same technologies: HTML forms, JavaScript, CSS and ReactJS and React Bootstrap.

5.2.1 Microservice 2: Database

5.2.1.1 HTML Forms

HTML forms are used when the user needs to insert information. This is what we referred in the *Backend Design* section as a "template". Play framework supplies Java form helpers that allow for an easier association between the form being filled out and the object and method associated with it (Figure 47). These forms are written in JavaScript, and the CSS is included in the same HTML file.

As it was previously mentioned in *Backend Design*, the UPDATE method was written using the CREATE request. This was due to the fact that we used HTML forms to introduce the data. Some requests are "forbidden" by HTML5, UPDATE being one of them. To be able to still use HTML form so that the User Interface remained accessible and effective, we decided to change slightly how the request worked, achieving essentially the same result.

```
@helper.form(action = routes.DBController.createPatient()){
<h1> Please insert new Patient Information</h1>
<label>UserID</label><br>
<input type="text" name="userID"><br>
<label>First Name</label><br>
<input type="text" name="firstName"><br>
<label>Last Name</label><br>
<input type="text" name="lastName"><br>
<label>Weight in kg</label><br>
<input type="number" step="0.01" name="weight"><br>
<label>Height in cm</label><br>
<input type="number" step="0.01" name="height"><br>
<label>Date of Birth yyyy-mm-dd format</label><br>
<input type="text" name="dob" placeholder="yyyy-MM-dd" min="1997-01-01" max="2030-12-31"><br>
<button class="button" type="submit">Send information</button>
```

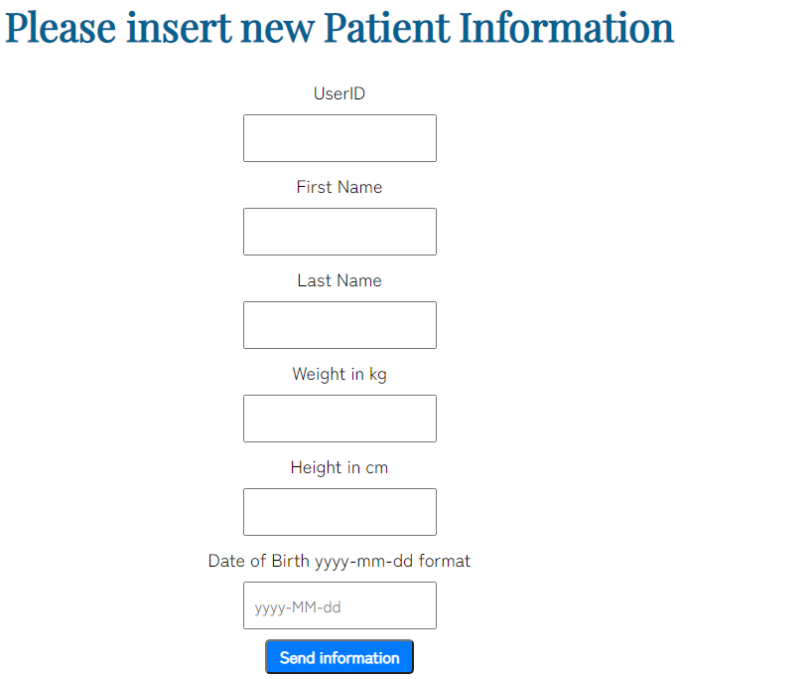


Figure 47 HTML form for the request POST /patients

The nature of HTML forms requires to use requests known as *implicit*. These requests however would cause the application to be non-RESTful. To avoid this, we can use JavaScript in the HTML form to directly open an explicit request with the information input in the form, as shown in Figure 48. As it can be seen, a new window will open with the explicit request when the *submit* event occurs.

```
<script type="text/javascript" language="javascript">
function redirect(event)
{
    var element = document.getElementById("form").elements["userID"].value;
    window.open('http://localhost:9000/patients/' +element);
}
</script>
<form id="form" onsubmit="redirect(event);window.close()">
    <h1> Please insert userID you want to search </h1>
    <label>UserID</label><br>
    <input type="text" name="userID"><br>
    <button class="button" type="submit">Submit</button>
</form>
```

Figure 48 Form passing information to the Controller method

5.2.2 Microservice 1: Expert System

5.2.2.1 FrontendController

FrontendController only has one method which reads the file *index.html*, which has the information to load the frontend part of the application, shown in Figure 49. This controller is written in Scala, as it has a simpler way of reading HTML files.

```
def index: Action[AnyContent] = assets.at( file = "index.html")
```

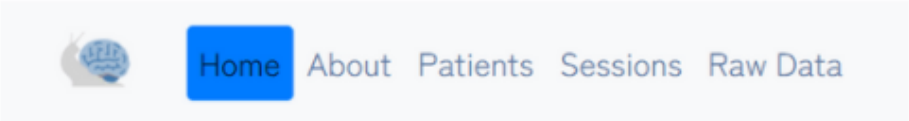
Figure 49 Method in FrontendController

5.2.2.2 ReactJS and React Bootstrap

These are libraries that allow the building of a web page using built-in classes that are more intuitive than JavaScript. They help when building certain components such as navigation bars, carousels, and cards.

The webpage was built using the wireframe explained previously as reference. React was used to build the main design components such as the navigation bar shown in Figure 50. The elements between <> such as <Navbar.Brand> indicate when a React component begins. The backward slash \ indicates when a component ends. If a component has more elements such as *bg = "light"*, it refers to its design or alignment

inside the page. These may also be edited using CSS, but for simple retouches the React components are normally accessed directly. All JavaScript files such as this one have an associated CSS file where elements such as font, colour and alignment can be thoroughly edited.




```
import React from 'react';
import './Navbar.css';
import { Navbar, Nav, Form, Button } from 'react-bootstrap';
import { withRouter } from 'react-router-dom';

const Navigation = (props) => {
    const { location } = props;
    return (
        <Navbar className="mb-0" bg="light" variant="light">
            <Navbar.Brand href="#home" >
                <img
                  alt=""
                  src={require ("./exoLogo.png")}
                  width="50"
                  height="auto"
                  className="d-inline-block align-top"
                />{' '}
            </Navbar.Brand>
            <Nav activeKey={location.pathname} variant = "pills" bg="light"
text="dark" className="ml-3">
                <Nav.Link href="/">Home</Nav.Link>
                <Nav.Link href="/about">About</Nav.Link>
                <Nav.Link href="/patientsUi">Patients</Nav.Link>
                <Nav.Link href="/sessionsUi">Sessions</Nav.Link>
                <Nav.Link href="/rawDataUi">Raw Data</Nav.Link>
            </Nav>
        </Navbar>
    )
}

export default withRouter(Navigation);
```

Figure 50 Example of code written using React

5.2.3 The User Interface

The first page shown when the application starts is the homepage, shown in Figure 51.

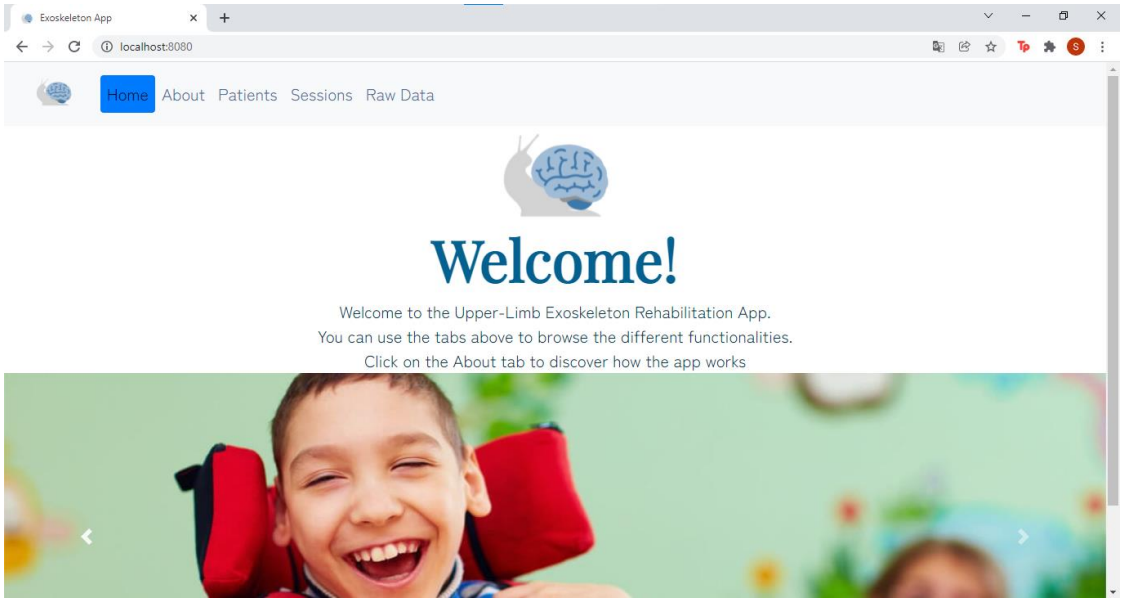


Figure 51 Homepage of the App

The About page (Figure 68, in the ANNEX) shows a summary of how the application works and what each section can do. The patients, sessions, and raw data pages (Figure 69 in the ANNEX, Figure 52 and Figure 70 also in the ANNEX, respectively) all show a set of cards that list the operations that can be done with those resources. These cards will open a new window that will carry out the operation, and that will close when the process has finished.

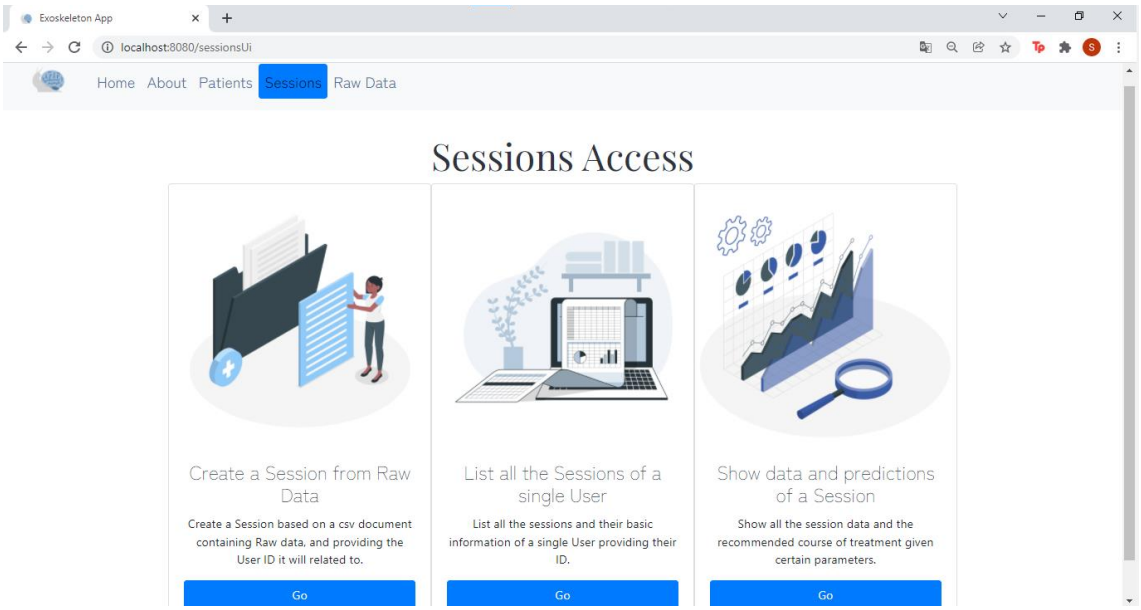


Figure 52 Sessions Page

To insert new information in the database, forms like the one shown in Figure 47 are used. Most of these forms are search bars like the one shown in Figure 53.

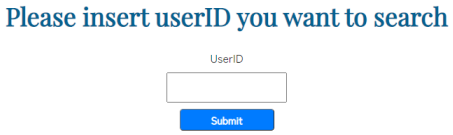


Figure 53 Example of a Search Bar

However, sometimes data has to be visualised, as shown in Figure 54.

Figure 54 Show Patient (left) and show Raw Data (right)

The validation of certain aspects of the application, such as form insertion or session calculations will be done in the next section.

6 TESTING AND RESULTS

This section will show how the different operations are carried out in the application. We will begin by showing how each microservice works: firstly how the Cassandra database works on its own, then how Drools works with the data from the database and the user interface.

6.1 Microservice 2: Accessing the Cassandra Database

The Cassandra database can be accessed directly or through the user interface. To use Cassandra, you must use *cqlsh*, its own command-line interface. It is similar to SQL and uses commands such as SELECT and INSERT. For example, if you wanted to access a certain session of a certain patient, you would execute the command shown in Figure 55. The equivalent of this command in the user interface is shown in the next section in Figure 58.

```
cqlsh:upperlimb> SELECT * FROM sessions WHERE userid='user001' AND sessionid='id1';

 userid  | sessionid | muscact1 | muscact2 | muscact3 | muscact4 | mvc1        | mvc2
---------+-----------+----------+----------+----------+----------+-------------+-----------
--
 user001 |       id1 | 28.57831 | 36.40171 | 18.93118 |  16.0888 | 88654.11719 | 1.037e+05
_


 mvc3         | mvc4         | rms1        | rms2        | rms3       | rms4
--------------+--------------+-------------+-------------+------------+----------

 16364.98828 | 31935.87695 | 2701.90918 | 5187.09277 | 831.62506 | 1054.2845
```

Figure 55 Command in *cqlsh* to access the Session

Although Cassandra can be used directly, inserting and selecting data, it is more complicated than the user interface. The results of the previous example are obtained using the calculations explained in section 5.1.2, and how it is done in the user interface is shown in Figure 56 How a Session is created using the User Interface The .csv files used where obtained through the *mDurance* application, and contain all the EMG values of a single session from a single patient. These files contain the results of the tests carried out by Laura Blanco [28] .

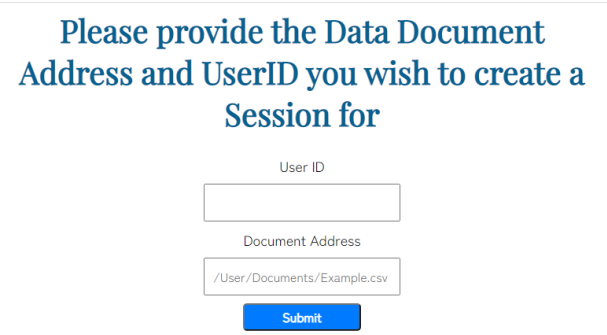


Figure 56 How a Session is created using the User Interface

6.2 Microservice 1: Session Prediction

This operation will be accessed by clicking on the button on the third card shown in Figure 52. Then, the form shown in Figure 57 will have to be filled out with the information of the rehabilitation session: the diagnosis of the patient, the number of elastics used during the session, and whether they were assisting or resisting flexion.

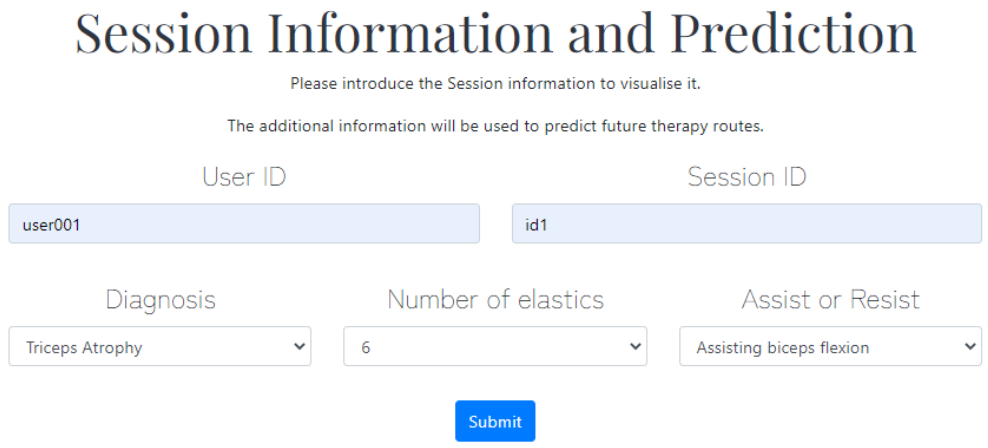


Figure 57 Form required for Session prediction

Once that information is submitted the following set of results will be displayed. The first part displayed are the calculated values for the muscular activity, muscle fibre recruitment and the maximum reference contraction in biceps and triceps (Figure 58). The recommendation will also be shown, explaining if the value of the muscular activity regarding the diagnosis is inside the correct margin. If it is not, it will recommend either to increase or decrease the number of elastics for the next session. If

it is inside the margin, it will recommend maintaining the number of elastics for the next session.

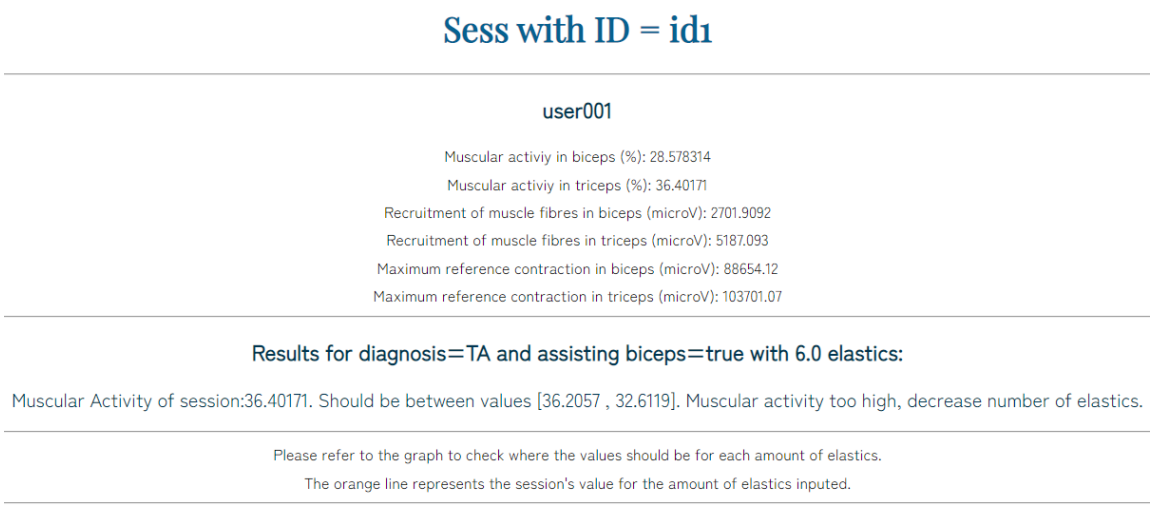


Figure 58 Results of prediction

The other part of the prediction shows a graph with the muscular activity of the session, the mean muscular activity, and the margin it should be in (Figure 59). This way, the therapist can refer to the graph to visually understand how far from the margin the session value was. The orange star represents the session value, when looking at its value, it can clearly be observed that it is higher than it should be. We also included the possibility to zoom into the graph by simply scrolling (Figure 60). For each diagnosis, assisting/resisting combination and number of elastics, there is a graph, so the graph will change with the input parameters.

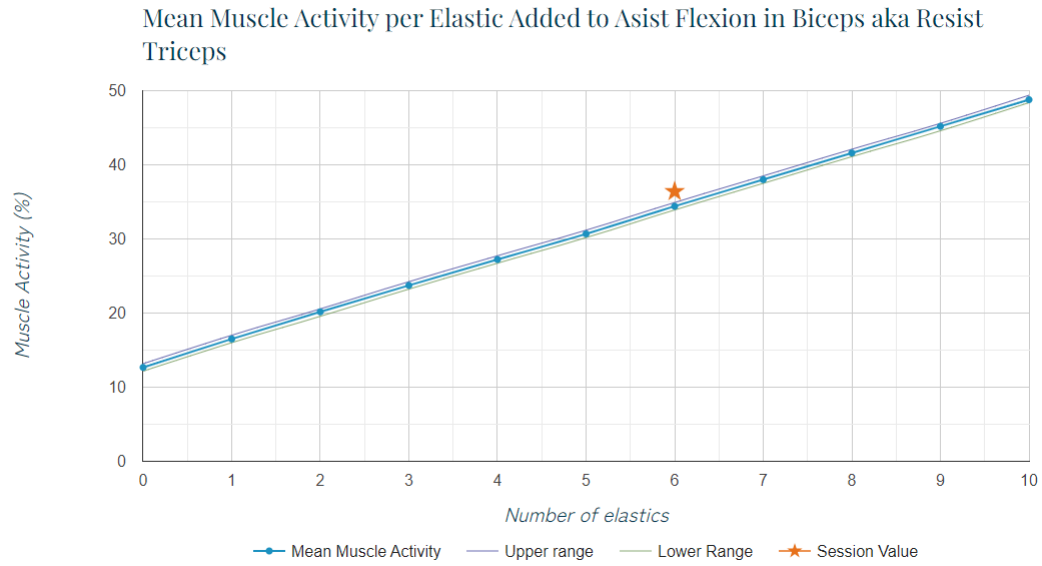


Figure 59 Graph shown with the predictions

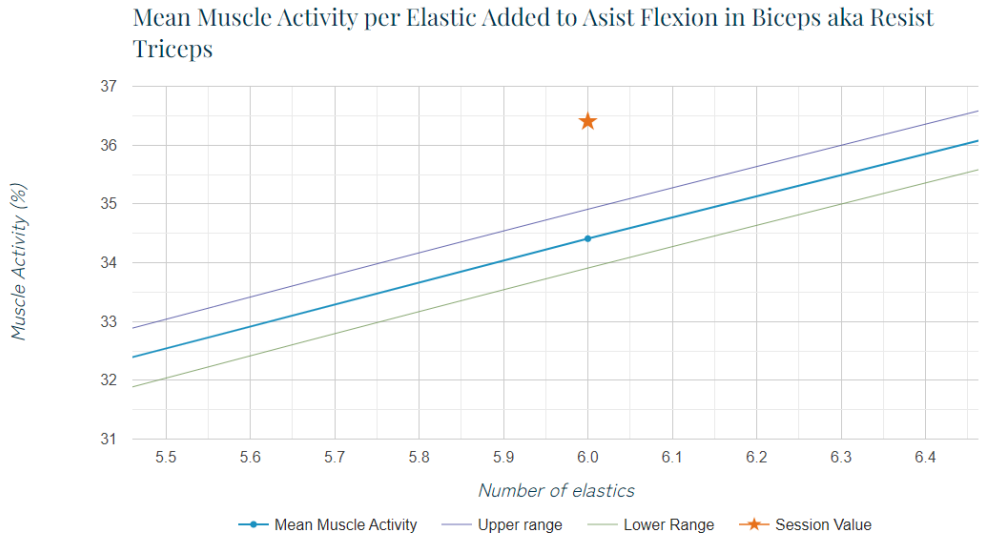


Figure 60 Zoomed-in graph by scrolling and dragging.

The graph has been coded using *Google Charts*, and the Excel spreadsheets explained in a previous section.

6.3 Request Management

In this section the different request codes (previously mentioned in the *Backend Design* section) and management will be explained.

Both of these requests are managed in the same way, showing a message box that will automatically close the window the user is at once they click the accept button. The code differs when the Result in the request has been returned an *ok* (200) or a *badRequest* (400) as shown in Figure 61.

```
        return ok(message.render("Patient created/updated correctly"));
return badRequest(messageCancel.render("Check the form for errors/missing values. " +
        "Click accept to close window. Click cancel to edit."));
```

Figure 61 Result ok (200, above) and Result badRequest (400, below)

These will show different message boxes, as shown in Figure 62.

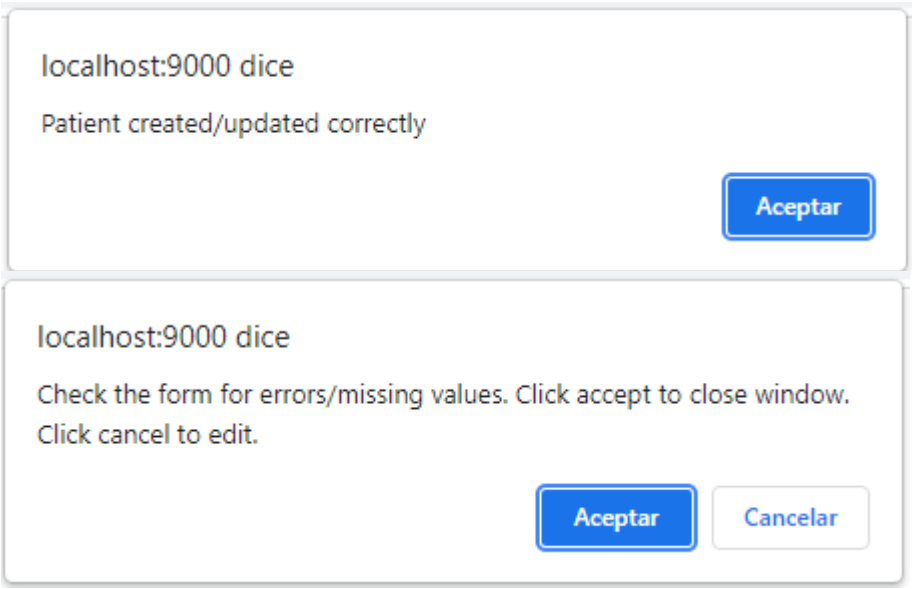


Figure 62 Message boxes for a code 200 (above) and a code 400 (below)

7 CONCLUSIONS

In this project, a microservices software architecture for an application designed to aid the rehabilitation process of patients with Cerebral Palsy was created. This application contains two microservices, implemented with Play framework and following a RESTful architecture. To create the architecture, we had to first understand what data could be obtained during the rehabilitation sessions. Given the large number of data recorded during each session, we realised that we had to use aggregated data for the predictions, as well as a way to store and retrieve this data efficiently. In [28], it is explained which values are useful to the therapist (muscular activity), so these were the values we decided to use.

These values are calculated in our database microservice. They are determined using R, which we had difficulty implementing with Java at first since it required that we use certain libraries to calculate the muscular activity and the result be stored in a Java object. This microservice manages all access to the Cassandra database. Most URIs of the application generate requests to access the database, such as creating a patient or viewing a session. The tables in the database were created keeping in mind how it would be more likely accessed. This determined how the application is divided, and how the therapist interacts with the user interface.

The other microservice we configured was the Rule-Based expert system, using the Drools engine. This part of the application is in charge of making decisions based on the data the physiotherapist introduces, as well as the data obtained during a rehabilitation session. It interacts with the Cassandra database, taking the Session data to compare it to values and reach conclusions, which means that we had to establish a connection between both. This meant that when the user sent a request to the Drools microservice, this one will send another request to the Cassandra microservice to obtain the data it needs.

To be able to clearly see the architecture we designed in action, a user interface was created using JavaScript and HTML. This was decided since the implementation of the application with the videogames could not be carried out, and therefore we could provide a user-friendly interface to better understand how it could work when the game

is included. As future work, apart from offering HTML representation of our data, JSON could also be given as an option so that the game could use it. This would require that each request indicates which representation is desired, either HTML or JSON. As it was explained in the Section 1, the use of games during the rehabilitation of Cerebral Palsy patients has proved to be very beneficial. This application could also be used in the case of using a game during rehabilitation. There would only have to be more parameters added, such as starting level or game of choice. These parameters will then be included in the rules as the others have been, and the predictions will be made taking into consideration the game parameters.

8 USER MANUAL

To use the application, you must first make sure that Cassandra is running. To do this, open a command line prompt in the *bin* directory of the Cassandra file. Then type "cassandra" and enter (Figure 63).

```
C:\Users\anaco\Desktop\apache-cassandra-3.11.10-bin\apache-cassandra-3.11.10\bin>cassandra
Detected powershell execution permissions.  Running with enhanced startup scripts.
```

Figure 63 Starting Cassandra

Once Cassandra is running, open up the code of each microservice in an IDE. Using the IDE terminal, make sure the directory corresponds to the project (drools-upperlimb for MS1 and cassandra-upperlimb for MS2).

Begin with starting MS2 by typing *sbt run*. This will automatically load the microservice in the port 9000 (Figure 64).

```
C:\Users\anaco\Documents\TFG\play-samples-play-java-starter-example\cassandra-upperlimb>sbt run
[info] welcome to sbt 1.3.13 (Oracle Corporation Java 1.8.0_251)
[info] loading global plugins from C:\Users\anaco\.sbt\1.0\plugins
[info] loading project definition from C:\Users\anaco\Documents\TFG\play-samples-play-java-starter-examp
[info] loading settings for project cassandra-upperlimb-build from plugins.sbt ...
[info] loading project definition from C:\Users\anaco\Documents\TFG\play-samples-play-java-starter-examp
[warn] There may be incompatibilities among your library dependencies; run 'evicted' to see detailed evi
[info] loading settings for project root from build.sbt ...
[info] set current project to proyectosI (in build file:/C:/Users/anaco/Documents/TFG/play-samples-play-

--- (Running the application, auto-reloading is enabled) ---

SLF4J: Class path contains multiple SLF4J bindings.
SLF4J: Found binding in [jar:file:/C:/Users/anaco/AppData/Local/Coursier/cache/v1/https/repo1.maven.org/
SLF4J: Found binding in [jar:file:/C:/Users/anaco/AppData/Local/Coursier/cache/v1/https/repo1.maven.org/
SLF4J: See http://www.slf4j.org/codes.html#multiple_bindings for an explanation.
SLF4J: Actual binding is of type [ch.qos.logback.classic.util.ContextSelectorStaticBinder]
[info] p.c.s.AkkaHttpServer - Listening for HTTP on /0:0:0:0:0:0:0:0:9000

(Server started, use Enter to stop and go back to the console...)
```

Figure 64 Starting MS2

In the case of running *Cassandra* on different machines, the contact point must be changed, both in the *.yaml* file and in the *additional.conf* file in the conf directory (Figure 65). Otherwise, CqlSession will connect directly to the default contact point and port.

```
cassandra {
    contact-point = "127.0.0.1"
}
```

Figure 65 additional.conf file

Next, start MS1 by typing *sbt "run 8080"*. This will start this microservice in the port 8080 (Figure 66).

```
C:\Users\anaco\Documents\TFG\playDrools\drools-upperlimb>sbt "run 8080"
[info] welcome to sbt 1.3.13 (Oracle Corporation Java 1.8.0_251)
[info] loading global plugins from C:\Users\anaco\.sbt\1.0\plugins
[info] loading settings for project drools-upperlimb-build from plugins.sbt ...
[info] loading project definition from C:\Users\anaco\Documents\TFG\playDrools\drools-upperlimb\project
[info] loading settings for project root from build.sbt ...
[info] set current project to drools-upperlimb (in build file:/C:/Users/anaco/Documents/TFG/playDrools/
[warn] insecure HTTP request is deprecated 'http://repository.jboss.org/nexus/content/groups/public-jbo
InsecureProtocol(true)

--- (Running the application, auto-reloading is enabled) ---

[info] p.c.s.AkkaHttpServer - Listening for HTTP on /0:0:0:0:0:0:0:0:8080

(Server started, use Enter to stop and go back to the console...)
```

Figure 66 Starting MS1

To begin using the application, start any browser and type in *localhost:8080*. This wil direct you to the homepage of the application (Figure 67).

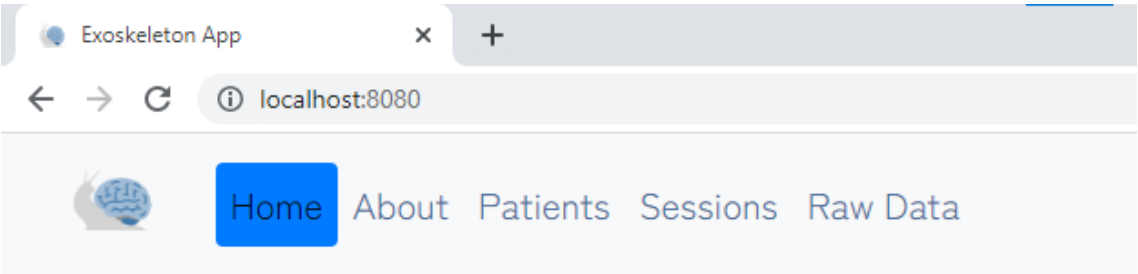


Figure 67 Starting the app

To close the application, simply close the browser. To stop the microservices from running, press enter on both IDE terminals. To stop the Cassandra cluster, press enter on the Command Line prompt.

9 REFERENCES

[1] S. Gulati and V. Sondhi, «Cerebral Palsy: An Overview,» *Indian Journal of Pediatrics,* 2018.

[2] «Centers for Disease Control and Prevention,» [Online]. Available: https://www.cdc.gov/ncbddd/cp/facts.html#..

[3] D. Patel, M. Neelakantan, K. Pandher and J. Merrick, «Cerebral Palsy in Children: a clinical overview,» *Transaltional Pediatrics,* vol. 9, nº 1, 2020.

[4] S. Korzeniewski, J. Slaughter, M. Lenski, P. Haak and N. Paneth, «The complex aetiology of cerebral palsy,» *Nature Reviews Neurology,* 2018.

[5] M. Sadowska, B. Sarecka-Hujar and I. Kopyta, «Cerebral Palsy: Current Opinions on Definition, Epidemiology, Risk Factors, Classification and Treatment Options,» *Neuropsychiatric Disease and Treatment,* 2020.

[6] «Cerebral Palsy Guide,» [Online]. Available: https://www.cerebralpalsyguide.com/cerebral-palsy/causes/risk-factors/. [Last access: May 2021].

[7] «Surveillance of cerebral palsy in Europe: a collaboration of cerebral palsy surveys and registers,» *Developmental Medicine & Child Neurology,* 2000.

[8] R. Palisano, P. Rosenbaum, S. Walter, E. Wood and B. Galuppi, «Development and reliability of a system to classify gross motor function in children with cerebral palsy,» *Develomental Medicine & Child Neurology,* 1997.

[9] C. Chukwukere Ogoke, «Clinical Classification of Cerebral Palsy,» *InTechOpen,* 2018.

[10] «Centers for Disease Control and Prevention,» [En línea]. Available: https://www.cdc.gov/ncbddd/cp/facts.html#.

[11] «Cerebral Palsy Alliance,» [Online]. Available: https://cerebralpalsy.org.au/our-research/about-cerebral-palsy/what-is-cerebral-palsy/severity-of-cerebral-palsy/gross-motor-function-classification-system/. [Last access: May 2021].

[12] T. M. O'Shea, «Diagnosis, Treatment, and Prevention of Cerebral Palsy,» *Clinical Obstetrics and Gynecology*, vol. 51, nº 4, 2008.

[13] D. L. Damiano, «Activity, Activity, Activity: Rethinking Our Physical Therapy Approach to Cerebral Palsy,» *Physical Therapy and Rehabilitation Journal*, vol. 86, nº 11, 2006.

[14] B. Bonnechère, L. Omelina, B. Jansen and S. Van Sint Jan, «Balance improvement after physical therapy training using specially,» *Diability and Rehabilitation*, 2015.

[15] H. Roberts, A. Shierk, N. Clegg, D. Baldwin, L. Smith, P. Yeatts and M. Delgado, «Constraint Induced Movement Therapy Camp for Children with Hemiplegic Cerebral Palsy Augmented by Use of an Exoskeleton to Play Games in Virtual Reality,» *Physical & Occupational Therapy In Pediatrics*, vol. 41, nº 2, 2021.

[16] J. Carpenter and E. Heweitt, Cassandra. The Definite Guide. Distributed Data at Web Scale., O'Reilly, 2016.

[17] N. Jatana, S. Puri, M. Ahuja, I. Kathuria and D. Gosain, «A Survey Comparison of Relational and Non-Relational Database,» *International Journal of Engineering Research & Technology*, 2012.

[18] M. S. Brown, «Get the basics on NoSQL databases: Wide,» *Forbes Magazine*, 2018.

[19] D. G. Borrow, S. Mittal and M. J. Stefik, «Expert Systems: Perils and Promise,»

Communications of the ACM, vol. 29, nº 9, 1986.

[20] V. Nagori and B. Trivedi, «Types of Expert System: Comparative Study,» *Asian Journal of Computer and Information Systems,* vol. 2, nº 2, 2014.

[21] W. Siler and J. Buckley, Fuzzy Expert Systems and Fuzzy, WILEY, 2005.

[22] M. Negnevitsky, Artificial Intelligence: A Guide to, Pearson, 2002.

[23] L. R. Mesker, Hybrid Neural Networks and Expert Systems, Springer Science and Business Media, 1994.

[24] «IBM: What is a Web Service?,» [Online]. Available: https://www.ibm.com/docs/en/cics-ts/5.2?topic=services-what-is-web-service. [Last access: 2021].

[25] L. Richardson and S. Ruby, RESTful Web Services, O'Reilly, 2007.

[26] R. Fielding, «Architectural Styles and the Design of Network-based Software Architectures.,» *Doctoral Dissertation,* 2000.

[27] «mDurance,» [Online]. Available: https://mdurance.eu/electromiografo-mdurance/. [Last access: 2021].

[28] L. Blanco Coloma, «Electromyography Validation of an Upper Limb Exoskeleton for Children with Cerebral Palsy,» 2021.

[29] «Drools Official Website,» [Online]. Available: https://drools.org/. [Last access: 2021].

[30] F. Curbera, M. Duftler, R. Khalaf, W. Nagy, N. Mukhi and S. Weerawarana, «Unraveling the Web Services web: an introduction to SOAP, WSDL, and UDDI,» *IEEE Internet Computing,* vol. 6, nº 2, 2002.

[31] JBoss Team, «Drools Expert User Guide,» [Online]. Available: https://docs.jboss.org/drools/release/6.0.0.CR3/drools-expert-docs/html_single/#d0e587. [Last access: 2021].

10 ANNEX

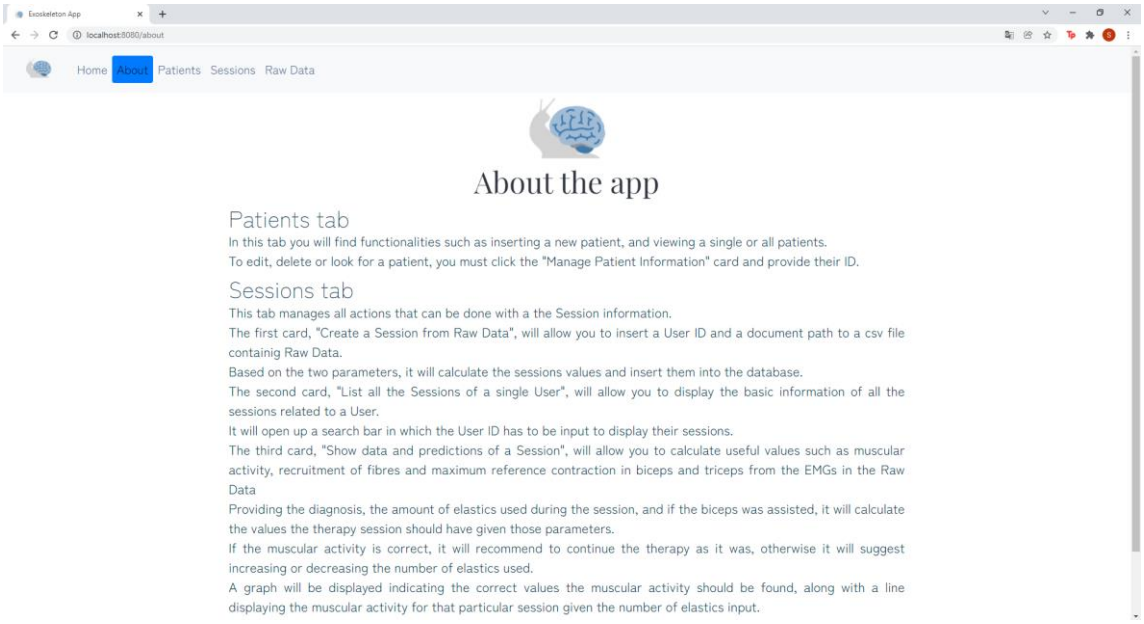


Figure 68 About Page

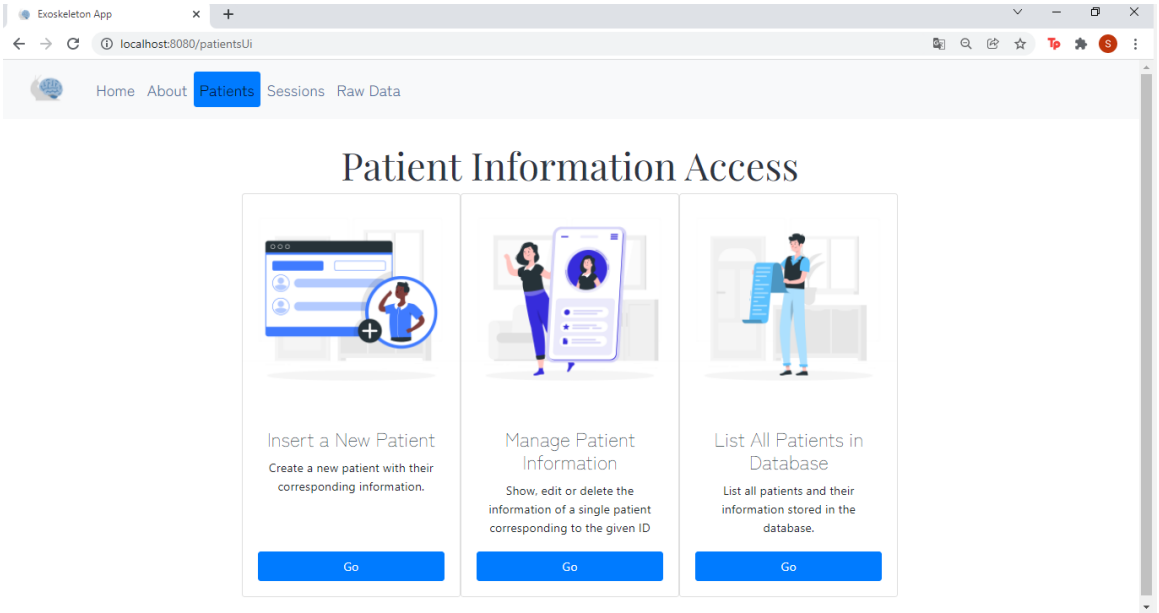


Figure 69 Patient Page

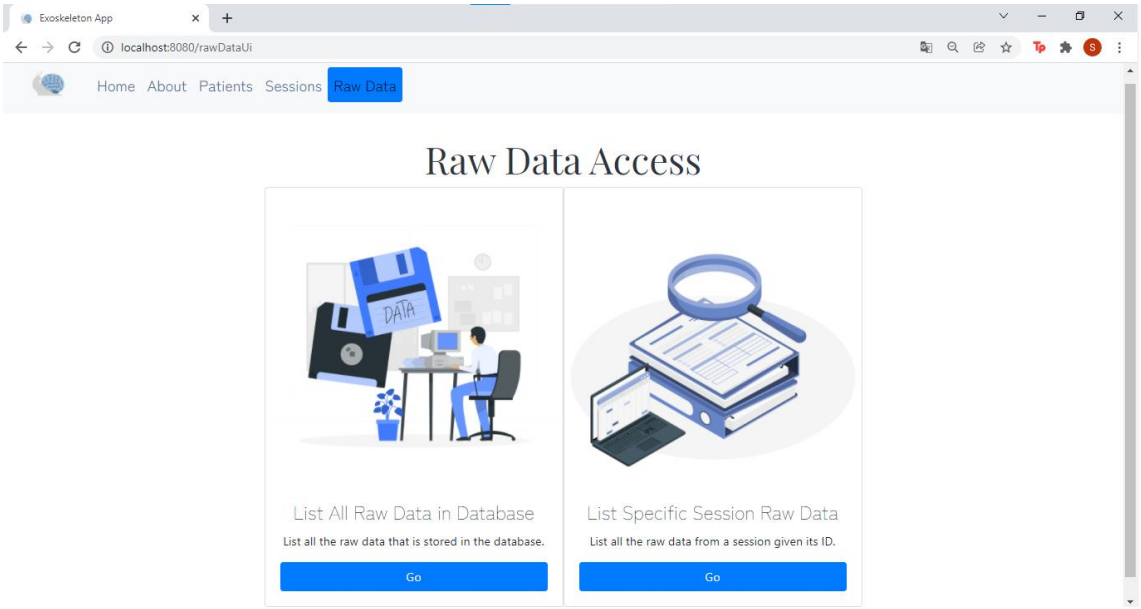


Figure 70 Raw Data Page