

UNIVERSITÀ DEGLI STUDI DI PADOVA

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

TESI DI LAUREA MAGISTRALE IN BIOINGEGNERIA



DEVELOPMENT OF AN EXTENSIBLE CLOUD APPLICATION FOR THE
MANAGEMENT, ANALYSIS, AND REMOTE SHARING OF
ELECTROENCEPHALOGRAPHIC DATA, COMPATIBLE WITH MICROMED
BRAIN QUICK SOFTWARE



Relatore

Dott. Ing. Marco Castellaro

Candidato

Antonio Marittimi

Matricola 2038150

Correlatore

Prof. Giovanni Sparacino

Dott. Ing. Raffaele Orsato

Dott. Alberto Pellizzon

A.A. 2023/2024

Contents

Contents

Contents.....	iii
Abstract:	v
1. Electroencephalography	1
1.1 The electroencephalogram.....	1
1.2 EEG signal	2
1.3 Recording system and electrode placement.....	5
1.4 Derivations	10
1.5 Electrodes montages.....	12
1.6 Clinical applications	14
2. Brain Quick Software®	17
2.1 Graphic user interface	17
2.2 Main software features.....	19
3. Overview of cloud platforms for EEG management.....	22
3.1 Cloud computing models.....	24
3.2 CloudVeneto platform overview	27
3.3 Available solutions for EEG management in a cloud platform	28
4. Methods: platform design and implementation	38
4.1 Django core.....	38
4.2 Django components.....	39
4.3 Brain quick cloud architecture	47
4.4 Front-end functionality.....	62
5. Results: platform deployment in CloudVeneto.....	70
5.1 Client setup.....	70
5.2 Server setup	73
5.3 Worker setup	75
5.4 Validation and testing.....	76
6. Documentation	79
6.1 Superuser	79

6.2	User groups	81
6.3	Users.....	82
6.4	Analysis	83
7.	Conclusion and future directions	88
7.1	Virtual machines configuration.....	88
7.3	Deployment example	88
7.4	Future improvements	92
	Bibliography	98
	Acknowledgements	100

Abstract:

This thesis focuses on integrating technologies such as electroencephalography (EEG) and cloud computing in the development of an application built with Python and Django for sharing and analyzing EEG data. This application is designed to be compatible with Micromed Brain Quick software, extending the functionalities of these systems and enabling more efficient management of EEG data. The main objective is to provide a platform that facilitates the remote sharing of EEG data and patient information. Using the cloud computing paradigm, the thesis proposes an innovative approach to storing, accessing, and managing data through the web, resulting in increased flexibility and ease of access for healthcare professionals. This allows them to request specific analyses and access EEG data anytime and from anywhere. The developed system leverages the capabilities of the Python programming language and the Django framework to create an intuitive user interface and a scalable architecture. The application can invoke specific analyses, integrating new algorithms into the workflow and enabling distributed processing on worker machines. This approach contributes to improving the overall system efficiency, facilitating collaboration and knowledge sharing in the field of electroencephalography.

1. Electroencephalography

1.1 The electroencephalogram

Human brain is comparable to a machine which is constantly processing information coming from the external environment. We always need to take care of what happens in our space, and we do this by collecting data from our senses. All this information is then funneled inside our central nervous system (CNS), which is capable of processing tons and tons of data. The information flow from the outside to our CNS is ruled by action potentials, a phenomenon caused by chemical and electrical cellular activities that propagates among nervous cells.

The cellular activity can be recorded in terms of signals, which analysis is an indicator of physiological state of the subject.

Electroencephalography is the science concerning the recording and the analysis of bioelectrical signal coming from brain nervous cells, the Electroencephalogram (EEG). Data coming from brain is key in the field of neuro-disorders diagnostics and has many clinical applications, above all:

- Epilepsy diagnosis
- Patient monitoring during anesthesia surgery
- Brain tumor detection
- Quantification of deficits in brain activity
- Study of sleep phases
- Study of EEG rhythm under effect of drugs or meditation

German physiologist and psychiatrist Hans Berger (1873 - 1941) is credited with recording the first human EEG brainwaves in 1924, after previous experiments performed on dogs and rabbits starting from 1890 with Adolf Beck.

Berger is considered as the inventor of electroencephalograph, a device to record EEG signals, described by David Millet "as one of the most surprising, remarkable, and momentous developments in the history of clinical neurology".

Nowadays, the EEG recording is obtained by applying sensors, small metal discs called electrodes, on the scalp. The electrodes pick up and record electrical events occurring in the underlying cerebral cortex. The collected EEG signals are then amplified, digitized, and sent to a computer or mobile device for storage and data processing. A key feature of electroencephalography is that the exam is non-invasive, and thus can be executed on all patients, independently from their health state.

Traces are recorded with a temporal resolution in milliseconds [*ms*], allowing real time analysis. Normally, EEG signals are in the microVolt [μV] range and have different frequency components.

1.2 EEG signal

Analyzing brain signals with Fourier's Transform algorithm, it's possible to extrapolate the frequencies that compose EEG data and to identify several areas, with distinct amplitudes and characteristics along the cerebral cortex. Different frequencies correspond to different cerebral functions. Those listed below are the most relevant in clinical practice.

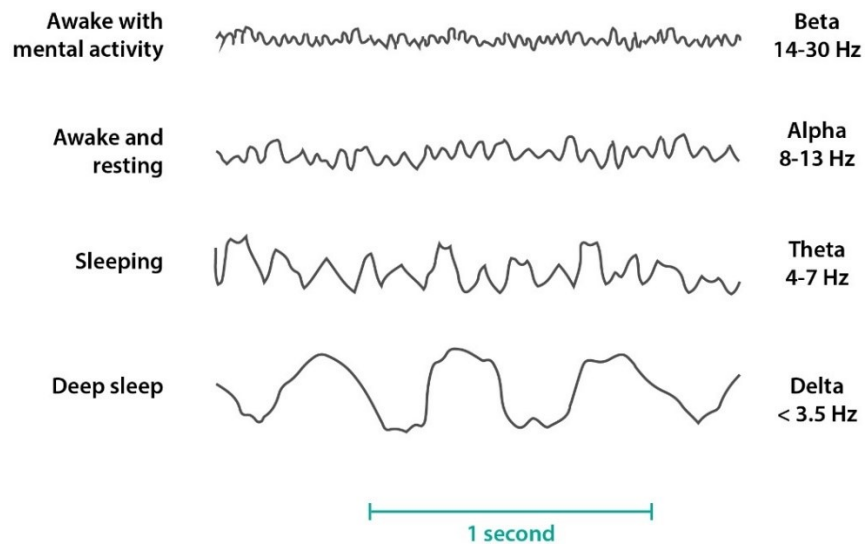


Figure 1: Main EEG components and related frequencies [3]

- **Beta (β) rhythm** (frequency range from 14 Hz to about 30 Hz), is divided into slow β (13.5-18 Hz) and fast β (18.5-30 Hz) and has an average electrical voltage of 19 μV (8-30 μV). They are most closely associated with being conscious or in an awake, attentive and alert state. Low-amplitude beta waves are associated with active concentration, or with a busy or anxious state of mind. Beta waves are also associated with motor decisions (suppression of movement and sensory feedback of motion).
- **Alpha (α) rhythm** (frequency range from 7 Hz to 13 Hz), is the basic rhythm present in EEG, also called "Berger rhythm"; it is possible to distinguish the slow α (8-9 Hz), the intermediate α (9-11.5 Hz) and rapid α (11.5-13 Hz), with an average amplitude of 30 μV . It is often associated with a relaxed, calm, and lucid state of mind. Alpha waves can be found in the occipital and parietal regions of the brain. They can be induced by closing one's eyes and relaxing, and they are rarely present during intense cognitive processes like thinking, mental calculus and problem-solving. In most adults, alpha waves range in frequency from 9 to 11 Hz.
- **Theta (θ) rhythm** (frequency range from 4 Hz to 7 Hz) detected in EEG measurement is often found in young adults, particularly over the temporal regions and during hyperventilation. It is divided into slow θ (4-6 Hz) and rapid θ (6-7.5 Hz), with an average voltage of 75 μV . In older individuals, theta activity with an amplitude greater than about 30 millivolts (mV) is seen less commonly and could be index of brain disorders. In normal conditions the theta phase occurs in the first minutes of falling asleep, when a subject is still in a state of drowsiness, where it is then alternated by graphemes called sleep spindles.
- **Delta (δ) rhythm** (frequency range from 0 Hz up to 4 Hz) are predominantly found in infants. Delta waves are associated with deep sleep stages in older subjects. They have been documented interictally (between seizures) in patients with absence seizures, which involve brief, sudden lapses in attention. Delta rhythms can be present during wakefulness — they are responsive to eye-opening and may be enhanced by hyperventilation as well. They are predominant in childhood, occur during general anesthesia of a subject, in some brain diseases or in general dysmetabolic diseases, such as hyperazotemia. Delta waves have an average voltage of 150 μV .

Other frequencies bands with clinical interest include:

- **Gamma (γ) rhythm**, a pattern of neural oscillation in humans with frequency between 25 and 140 Hz, correlated with large-scale brain network activity and cognitive phenomena such as working memory and attention.
- **High Frequency Oscillations (HFO)** are brain waves with frequency faster than 80 Hz, generated by neuronal cell population. They are present in physiological state during sharp waves and ripples - oscillatory patterns involved in memory consolidation processes. HFOs are associated with pathophysiology of the brain like epileptic seizure and are often recorded during seizure onset. It makes a promising biomarker for the identification of the epileptogenic zone [4].

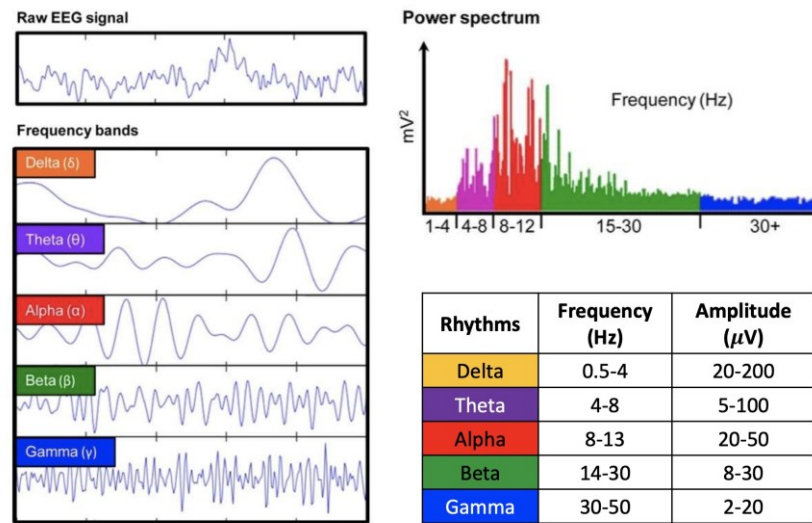


Figure 2: EEG frequencies components, from Delta to Gamma [5]

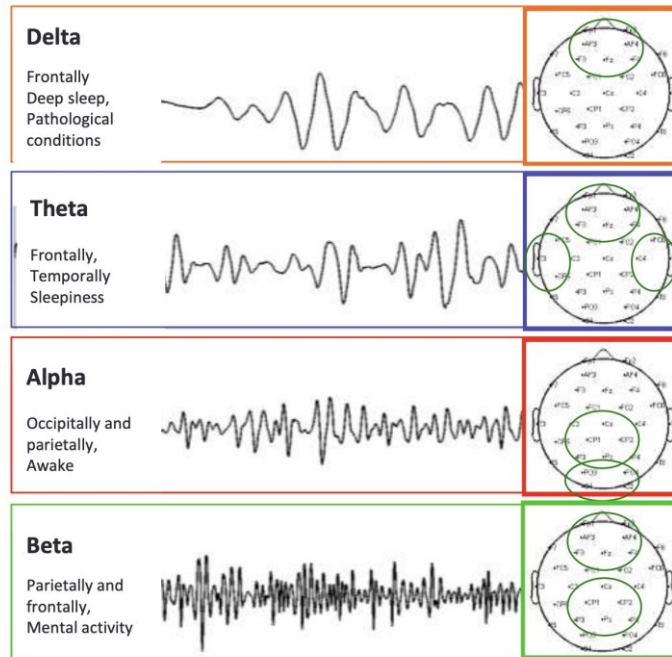


Figure 3: Spatial localization of EEG rhythms [5]

1.3 Recording system and electrode placement

The recording system for EEG signals, known as an electroencephalograph, comprises an acquisition module for gathering signals from the scalp through measurement electrodes integrated into a specially designed headset worn on the patient's head. Additionally, there is a signal processing module and a unit for displaying and storing the collected data.

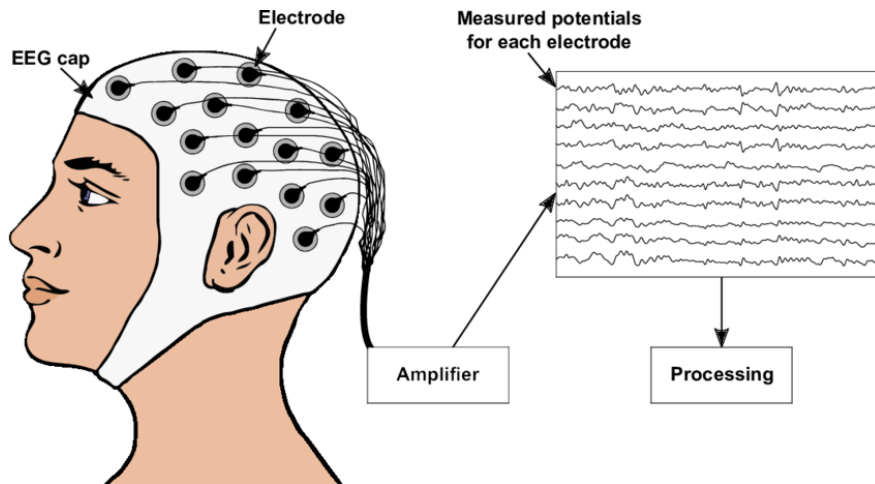


Figure 4: EEG recording system scheme [6]

Precise positioning of electrodes on the scalp is key to obtaining accurate derivations and maximizing the signal-to-noise ratio during recording. Herbert Jasper introduced the 10-20 system at the 1957 Brussels IV International EEG Congress, offering a standardized method for electrode placement. Widely acknowledged internationally, this approach uses anatomical landmarks to ensure consistent electrode positioning. Its foundation lies in establishing a correlation between electrode locations and specific areas of the cerebral cortex, thereby ensuring comprehensive coverage of all regions of the brain.

The numerical labels "10" and "20" in the 10-20 system correspond to the distances between neighboring electrodes, representing either 10% or 20% of the overall distance (front-back or right-left) across the skull. This total distance is determined by specific anatomical landmarks on the scalp: the nasion and inion guide the front-back direction, while the two preauricular points guide the right-left direction. Using these landmarks, electrode placement is calculated along these directions based on pre-defined proportions: 10% is measured from the anatomical landmarks for the first electrode in that direction, and 20% is used for the subsequent electrodes. For instance, Fp1 is positioned at 10% of the total distance from the nasion, and Fz is then located at 20% of the total distance from Fp1.

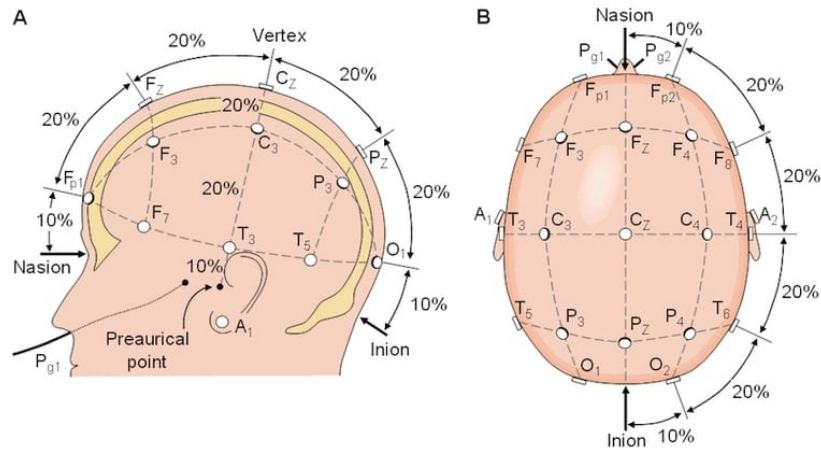


Figure 5: The 10-20 system with front-back (nasion to inion) 10% and 20% electrode distances [7]

The designation of each electrode is indicative of the general brain region it covers. When moving from the front to the back, the electrode letters are assigned as follows:

- Pre-frontal or frontal (*Fp*)
- Frontal (*F*)
- Central line of the brain (*C*)
- Temporal (*T*)
- Parietal (*P*)
- Occipital (*O*)

For electrodes situated between these lines, a combination of multiple letters is employed in an order from front to back. This primarily pertains to systems with higher electrode density. Additionally, the letters M and A are occasionally utilized to denote the mastoids or earlobes, respectively. These locations are typically included to serve as reference points for offline signal analysis.

In practical application of the SI 1020 method for electrode placement on the scalp, it is essential to draw precise lines from specific anatomical reference points. These foundational lines, perpendicular to each other, can be outlined as follows:

1. **Antero-posterior middle line** (Figure 6): this line connects the nasion (upper hairline of the nose) to the inion (prominence at the base of the occipital bone) while passing through the vertex. Along this line, the 5 standard positions are identified. To determine the placement of the fronto-polar point (Fpz) and occipital point (Oz), they are located at 10% of the total distance from nasion and inion, respectively. All other points along this line are calculated at 20% intervals between Fpz and Oz. The 10-20 system gets its name from this precise calculation of distances between electrodes. The middle electrode, as per the ideal arrangement, should be positioned exactly in the midpoint between nasion and inion.

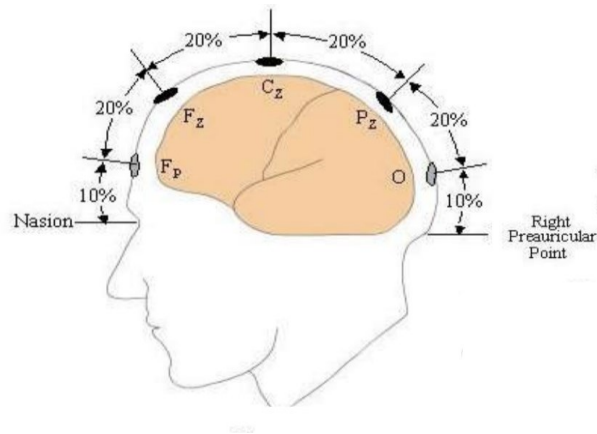


Figure 6: Electrodes placement representation on Anterior-posterior middle line, going from Fpz to Oz [5]

2. **Latero-lateral coronal line** (Figure 7): this line connects the right and left preauricular points through the central vertex point. Along this line, the temporal electrodes are placed at 10% of the total distance, starting from the preauricular point. The lateral central electrodes are then situated at 20% intervals from the temporal points and the median central point [5].

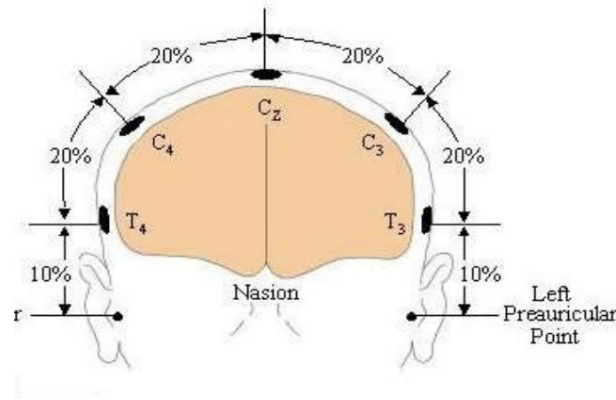


Figure 7: Electrodes placement representation on Latero-lateral coronal line, going from T4 to T3 [5]

Building upon the foundation of the perpendicular lines mentioned earlier, which help define electrode positions, we can determine the locations of electrodes aligned longitudinally alongside the middle line and those situated on the two coronal frontal and parietal lines. These lines extend respectively to the front and back of the coronal line that intersects the vertex. Frontopolar electrodes (Fp2 and Fp1) find their place along the longitudinal line, positioned at 10% of the lateral distance from Fpz. Conversely, for occipital electrodes (O1 and O2), the 10% calculation is made in relation to Oz. The inferior frontal (F8 and F7) and posterior temporal (T6 and T5) electrodes are positioned at 20% intervals along this line, originating from Fp/Fp1 to O1/O2. As for the remaining frontal (F4 and F3) and parietal (P4 and P3) electrodes, they are situated along the coronal frontal and parietal lines, maintaining an equal distance between the medial and temporal lines on each side. The 10-20 system quickly established a standard electrode placement on the scalp, facilitating consistent result comparison across laboratories globally. However, this method, not immune to criticism, has paved the way for higher-resolution application techniques, such as the 10-10 and 10-5 systems. [5]

The 10-10 system for instance, employing 81 electrodes, enables a more detailed identification of scalp localizations.

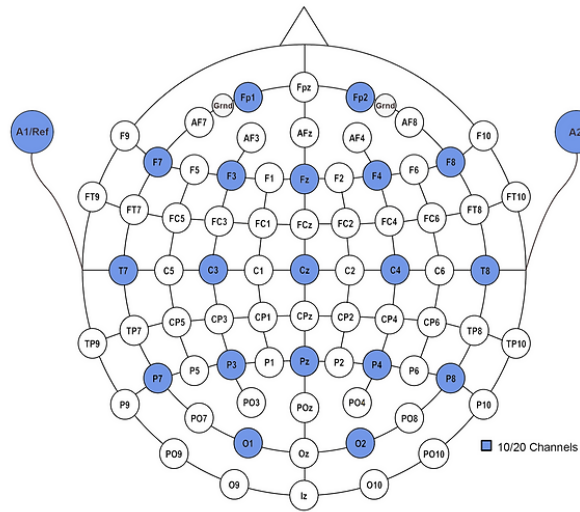


Figure 8: 10-10 electrodes placement, with reference to 10-20 channels [5]

1.4 Derivations

When examining the representation of the brain's bioelectric signal on an EEG trace, various factors come into play. Beyond the electrode placement on the scalp, the way these electrodes are linked to amplifiers is a critical aspect. In the realm of electroencephalography, the synergy between electrodes and their connection to the amplifier, involving mounts and leads, holds indispensable importance. This practice is rooted in historical significance and practical considerations. Typically, the EEG is portrayed as a series of traces, illustrating the dynamic changes in potential differences over time. In the traditional analog EEG setup, each trace originates from connecting two electrodes to amplifiers and filtering it. Subsequently, the captured signal travels through a galvanometer, an instrument designed for measuring small electrical currents, before being transcribed by a writing pen.

The evolution to digital EEG has seen a transition to computer hardware and software handling the entire process. Despite this shift, each trace, whether originating from an analog device or processed by a computer, retains its characterization as a channel. This persistent categorization emphasizes the

ongoing significance of tracing electrical activity for a comprehensive understanding of brain function.

The primary types of derivation in EEG recordings include:

1- Referential Derivations (Common Reference, Average Reference):

Common-Reference Mode: Each scalp electrode is referenced to a common electrode placed at a specific point, denoted as x. The challenge lies in finding a neutral common reference electrode, free from contamination by other electrical potentials and biological body potentials—an ideal but rare scenario. The major drawback of common referencing is referential contamination, wherein electrodes near a potential peak cause voltage change in all referenced electrodes. Equipotential electrodes with the reference reach zero, while those less engaged with the reference exhibit a pseudo-positive response. In theory, understanding the distribution of potential should be straightforward given a known electric field. However, in practice, the process is reversed.

Average-Reference Mode, also known as the mathematical reference, introduced in 1950 by Goldman and Offner, overcomes many common reference issues. It involves considering the average potential of all electrodes as the reference, allowing for a more stable and less contaminated baseline. The mathematical mean of a series of numerical values ensures that the sum of differences from the mean is zero, resulting in positive or negative deflections on the EEG trace relative to the zero value of the reference. [5]

2- Bipolar Derivations:

In bipolar derivations, potential differences are calculated between pairs of electrodes arranged along longitudinal or transversal chains. These chains share a common electrode between two successive channels. Consequently, an event beneath a specific electrode generates an equal but opposite deflection in the two adjacent electrodes preceding and following it in the electrode chain. This approach provides a localized perspective on potential changes between electrode pairs. Due to the highly variable nature of EEG patterns—ranging from focal to diffuse, transient to persistent—there isn't a single optimal derivation that can effectively capture all types of brain activity. One crucial consideration in lead selection is the

distance between electrodes, particularly in the context of common active and bipolar leads.

In bipolar leads, the paired electrodes have small and equal distances, accentuating the portrayal of rapid EEG activities. On the other hand, common active reference leads feature larger and unequal distances between electrodes, facilitating signal amplification and better highlighting slow activity. Illustrated in Figure 4, the visualization of a real epileptic focus undergoes changes when recorded with digital equipment, depending on the lead used—whether it's a bipolar lead, medium referential (AVG), or active common referential (G2). These variations in lead types contribute to nuanced representations of brain activity, showcasing the importance of thoughtful lead selection in EEG recordings.

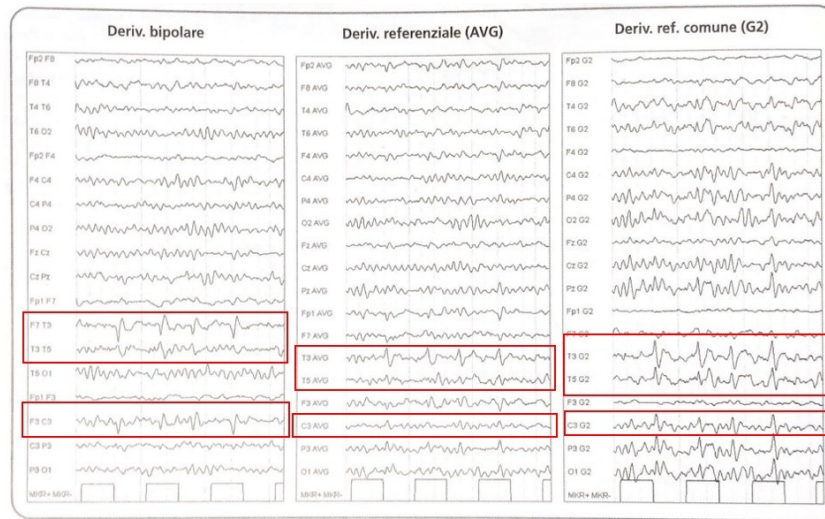


Figure 9: How real epileptic focus changes depending on the derivation visualized in bipolar and referential derivation. Seizure spikes are highlighted in red [5]

1.5 Electrodes montages

Talking about montages in electroencephalography concerns how electrodes are connected to the recording channel. EEG laboratories and technicians employ various montages for routine recordings, making it difficult to exchange

information among specialists. To address this, the International Federation of Clinical Neurophysiology (IFCN) and the American Clinical Neurophysiology Societies (ACNS) have issued fitting guidelines. The different assemblies are known as longitudinal bipolar (LB), transverse bipolar (TB), or referential (R), each designed for 16, 18, and 20 channels.

In summary, key recommendations include:

- Simultaneously record from at least 16 EEG channels.
- Position at least 21 electrodes following the 10-20 system.
- Utilize bipolar and referential assemblies.
- Clearly specify electrode connections at the start of each derivation using a simple and easily understood mode.
- In bipolar leads, maintain continuous lines with equal interelectrode distances.
- Ensure antero-posterior electrode progression.

Figure 5 shows routinely used bipolar montages based on the number of electrodes applied, considering the patient's head size. For common reference mounting, international guidelines suggest using the right auricle (A2) as a reference for right electrodes and the left auricle (A1) for left electrodes. In digital electroencephalography, G2 can also serve as an active reference, positioned on the midline anterior to Fz [5].

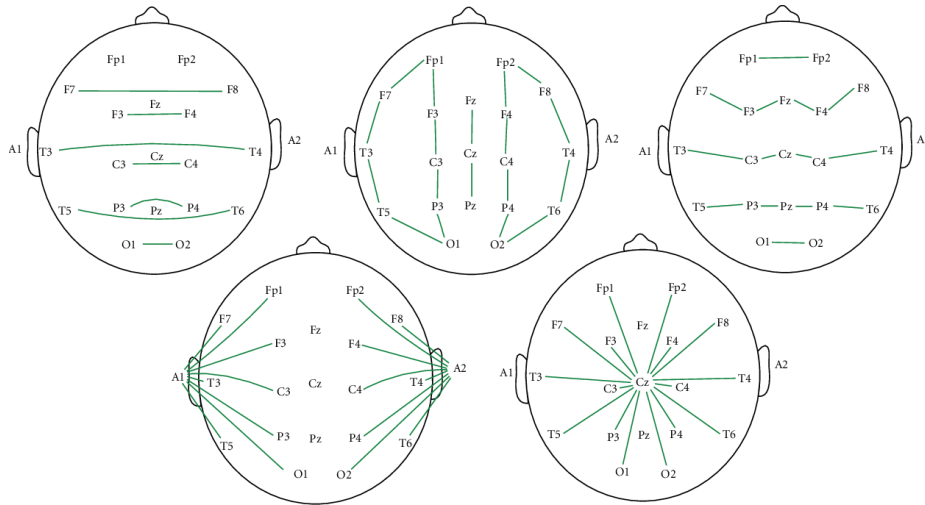


Figure 10: Spectral peaks montage maps. Lines correspond to subtractions used to calculate spectral peaks. From left to right, top to bottom: Counterpart Bipolar, Longitudinal Bipolar, Crossed Bipolar, Biauricular reference, and Cz reference [7]

1.6 Clinical applications

EEG stands as a fundamental diagnostic and monitoring tool within the realm of clinical neurophysiology, showcasing a wide range of applications for brain function analysis. Its significance is particularly pronounced in various clinical scenarios, with each application contributing uniquely to the understanding and management of neurological disorders. EEG is a non-invasive procedure, making it well-suited for individuals of all age groups, allowing doctors to observe and analyze brain activities without the need for invasive measures. Beyond its non-invasiveness, the EEG excels in its ability not only to record abnormal activities within the brain but also to pinpoint their specific locations.

One of the primary clinical roles of EEG lies in the diagnosis and classification of epilepsy. By capturing and analyzing abnormal electrical activity in the brain, such as epileptic spikes and waves, EEG aids in determining the specific type of epilepsy a patient may be experiencing, thereby guiding tailored treatment approaches. Beyond diagnosis, EEG assumes a critical role in the long-term monitoring (LTM) of patients with epilepsy, offering continuous observation to capture and characterize seizure activity. This meticulous process, often coupled with video recording, allows clinicians to correlate clinical events with their

corresponding electroencephalographic patterns. Moreover, EEG plays a key role in assessing patients experiencing altered states of consciousness. Whether individuals are in a coma or facing unexplained periods of confusion, EEG patterns become an instrument to comprehend the underlying neurological conditions contributing to these altered states.

In the realm of sleep medicine, EEG is a fundamental tool for evaluating various sleep disorders, uncovering abnormalities associated with conditions such as sleep apnea or parasomnias, contributing significantly to the field of sleep medicine.

During surgery, particularly in procedures involving the brain, real-time EEG monitoring has been proved as a valuable ally. It provides insights into the ongoing functionality of the patient's brain, particularly in surgeries where patients need to be awake or when there exists a risk of compromising critical brain regions.

The versatility of EEG extends to the evaluation of a spectrum of neurological disorders, including brain tumors, encephalopathies, and neurodegenerative diseases. By capturing distinctive patterns, EEG becomes a diagnostic tool, guiding clinicians toward the identification and understanding of underlying pathologies. Additionally, EEG finds application in the assessment of brain injuries, playing a crucial role in delineating the extent and location of traumatic brain injuries and other forms of cerebral damage.

Even in the domain of psychiatry, though less specific than in neurological disorders, EEG offers valuable insights. It aids in the assessment of certain psychiatric conditions, such as schizophrenia and mood disorders, where discernible patterns may hint at underlying brain dysfunction.

Beyond its clinical applications, EEG holds a prominent place in research settings, contributing to the study of brain function and cognitive processes. Researchers leverage EEG to explore the intricacies of memory, attention, perception, and various other cognitive functions. Furthermore, EEG biofeedback, also known as neurofeedback, emerges as a therapeutic avenue. This innovative approach employs real-time EEG data to train individuals in controlling specific brainwave patterns. Its applications span conditions such as attention-deficit/hyperactivity disorder (ADHD) and anxiety.

In essence, EEG emerges not merely as a diagnostic tool but as a comprehensive and indispensable asset in the clinical landscape, offering invaluable insights that shape diagnostic pathways and create tailored therapeutic strategies [1][2][5].

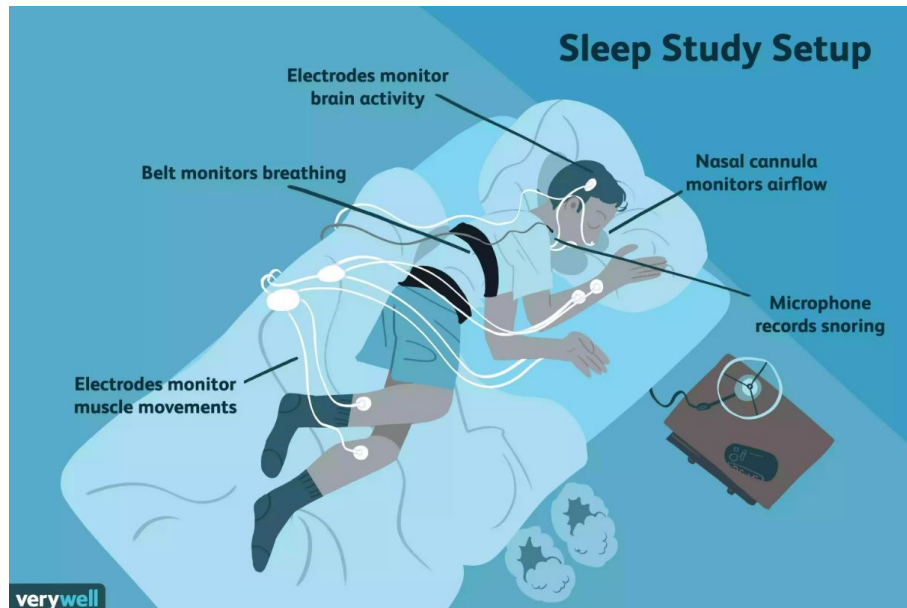


Figure 11: Example of a sleep study setup, with the main signals involved [8]

2. Brain Quick Software®

Brain Quick Software® stands as Micromed's leading product in the EEG market. This advanced software serves as a solution for managing various aspects of EEG and Video EEG processes, including acquisition, review, Long Term Epilepsy Monitoring (LTM), Stereo EEG, and Ambulatory EEG/PSG. Brain Quick Software® relies on **File Manager**, an intuitive archive interface that oversees both basic and advanced archive operations. Specifically, it facilitates fundamental patient and exam operations through Create, Read, Update, and Delete (CRUD) functionalities. Among its capabilities are the copying or moving of patients from one resource to another, as well as the transfer of exams between resources and patients/exams filtering. The software also automates patient migration from one resource to another post-acquisition and ensures the automatic archiving of patients after reporting. From File Manager, users are able to launch **Brain Quick Acquisition** directly, allowing to start the recording of a new Video EEG exam or to access existing Video EEG Exams through **Brain Quick Review**. The synergy between Brain Quick Software® and File Manager thus provides a user-friendly and comprehensive platform for EEG management.

2.1 Graphic user interface

The user can easily start File Manager using its dedicated icon. As soon as File Manager starts up, user is automatically projected in one of the available resources, displaying all existing patients.

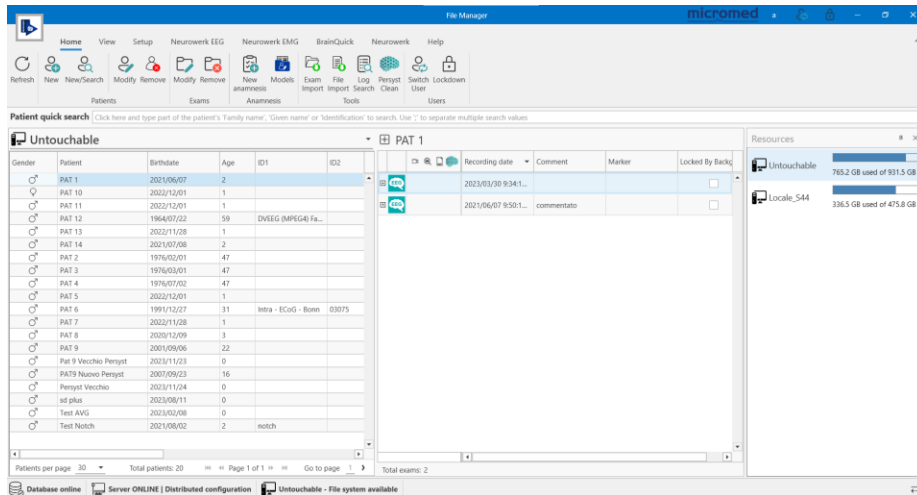


Figure 12: File Manager interface at startup [9]

In Figure 12, the organizational structure of File Manager is illustrated. In particular, the patients' list is featured on the left, providing a comprehensive view, while all available exams for each patient are presented on the right. The resources panel on the extreme right facilitates navigation among different resources. At the top section, File Manager's functionalities are arranged into tabs within a Ribbon Bar. Of particular interest is the Brain Quick Tab within this Ribbon Bar, situated on the upper part. This tab serves as a hub for various actions, allowing users to initiate a new EEG or Video EEG acquisition, create a new report, and access other advanced functions (refer to Figure 13).

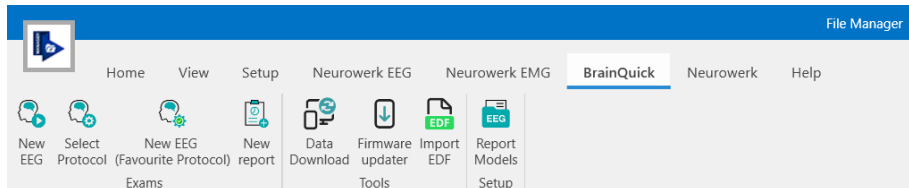


Figure 13: Brain Quick Tab with all available functionalities [9]

Always from archive view, double clicking on a EEG exam, Brain Quick is directly started, allowing to review the existing trace (Figure 14).

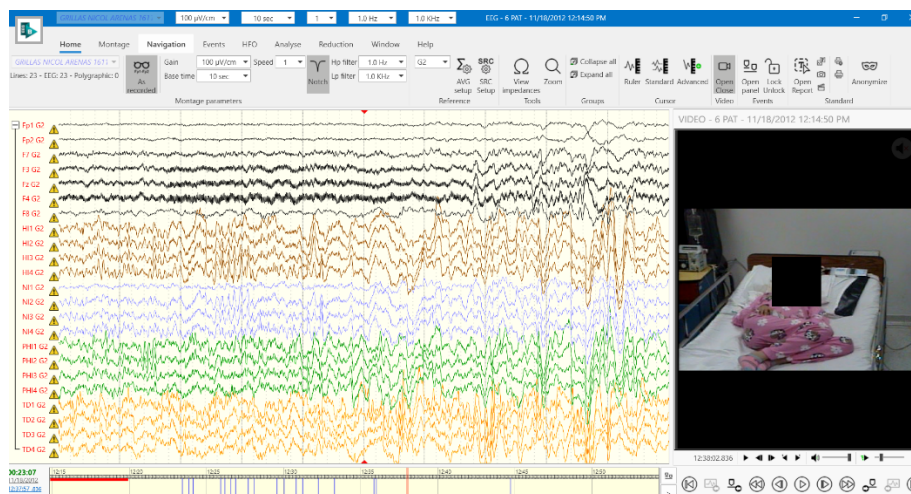


Figure 14: How Brain Quick Software appears reviewing an EEG [9]

The interface of Brain Quick Software presents a similar organization, containing a title bar from which it is possible to view patient's information, and a ribbon bar divided into tabs, which give access to all review functions. Both Brain Quick Software and File Manager offer the possibility to customize Ribbon Bar, defining desired functions to be present in each tab.

2.2 Main software features

Brain Quick Software is a comprehensive tool designed to support physicians in the recording, reviewing, and analysis of data obtained from Micromed digital acquisition systems. Its versatile application extends to EEG, LTM, PSG, useful for neurophysiological studies. For instance, the software is equipped for specialized tasks, such as cortical and photic stimulation during electroencephalography examinations (stereo EEG), in conjunction with specific Micromed stimulators. Furthermore, it offers the capability to monitor physiological measurements, including Intracranial Pressure (ICP), Brain Tissue Oxygen (PbtO₂), Cerebral Perfusion, Heart Rate (HR), and Blood Oxygen Saturation (SpO₂), sourced from interfaced third-party medical devices [9].

While the software provides predefined functional analysis tools, it is essential to emphasize that the results from these tools should not replace critical interpretation and clinical conclusions. Therefore, the use of Brain Quick Software is intended for qualified individuals, such as physicians, technicians, or healthcare professionals educated in biopotential recording, and serves more as a Clinical Decision Support System. Consequently, the utilization of BQ Software should always occur under the supervision of a physician or a qualified technician. The software supports international usage by offering multiple language options, accommodating characters from various languages, including non-character languages like Chinese and Norwegian. Functionally, BQ Software is designed to be multifaceted, enabling the management of archived and non-archived historical studies, remote reviews of ongoing studies performed by the SystemPlus Evolution acquisition system, report creation and template management, multiview examinations, initiation of new EEG recordings based on predefined or selected protocols, and creation of new histories and reports along with the management of report templates.

File Manager is able to work into 2 different operative conditions:

- **Standalone Environment:** machine serves both as a client and as a server
- **Distributed Environment:** a machine serves as a server, while all other machines in the system serve as clients. This configuration offers lots of advantages, such as settings centralization, central logs management, notifications management.

In terms of configuration, both Brain Quick Software and File Manager operate on three levels:

- **User Level:** Individual settings unique to a specific user in an environment, encompassing preferences such as color preferences, panels disposition, interface customization.
- **Unit Level:** Settings unique for all users on a specific machine, covering configurations like electrode positions, event definitions, average (AVG) reference configuration, specific paths and acquisition list configuration. Each unit setting has the possibility to be centralized, to be spread in all machines of the system.

- **Central Level:** Settings applied universally to all machines and users within the system, including aspects like Internationalization, Language, Notch configuration, File Manager labels.

3. Overview of cloud platforms for EEG management

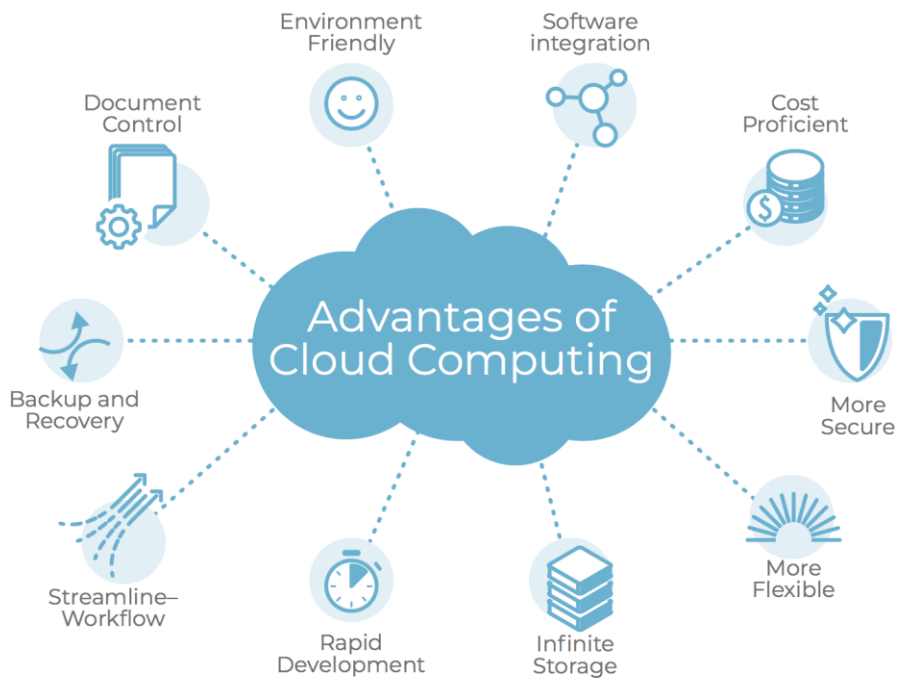
In the landscape of modern computing, the emergence of cloud platforms has transformed the way organizations manage and deploy their digital infrastructure. The essence of cloud computing lies in its ability to provide on-demand access to a myriad of computing resources over the internet. In this chapter, the goal is to dive in the world of cloud platforms and explore the diverse array of available solutions that define the contemporary market in EEG segment. The concept of a cloud platform, signifies more than a technological innovation: organizations, both large and small, are increasingly relying on cloud platforms to streamline processes, enhance scalability, and optimize resource utilization.

Cloud computing allows individuals and organizations to access computing resources over the internet, including applications, physical and virtual servers, data storage, development tools, and networking capabilities. These resources are hosted in a remote data center managed by a cloud services provider (CSP), who offers them through a subscription fee or usage-based billing. The term **cloud computing** also includes the technology that makes it work. This involves virtualized IT infrastructure, where servers, operating system software, networking, and other components are abstracted using special software. This abstraction allows pooling and division of resources, irrespective of physical hardware boundaries. For instance, a single hardware server can be divided into multiple virtual servers. Virtualization enables cloud providers to optimize the use of data center resources. Many corporations adopt the cloud delivery model for their on-premises infrastructure to achieve maximum utilization and cost savings compared to traditional IT. This model also provides self-service and agility to end-users.

Compared to traditional IT services, cloud computing provides several advantages:

1. **Cost Reduction:** Cloud computing allows you to offload the costs and efforts associated with purchasing, installing, configuring, and managing on-premises infrastructure.
2. **Improved Agility and Time-to-Value:** Organizations can quickly use enterprise applications, reducing the waiting time for IT responses, hardware setup, and software installation. Cloud empowers users, especially developers and data scientists, to access software and support infrastructure independently.
3. **Scalability:** Cloud offers elasticity, enabling the scaling of capacity based on demand. This avoids the need to purchase excess capacity during slow periods, and the global network of cloud providers helps distribute applications closer to users worldwide.

Whether at home or work, individuals likely engage in cloud computing daily through applications like Google Gmail, streaming media such as Netflix, or cloud file storage like Dropbox. According to industry analyst Gartner, global end-user public cloud spending is projected to reach nearly USD 600 billion in 2023 [10].



3.1 Cloud computing models

Cloud computing services come in three primary models: **Infrastructure-as-a-Service** (IaaS), **Platform-as-a-Service** (PaaS), and **Software-as-a-Service** (SaaS), and organizations often utilize a combination of these [10].

- **Software-as-a-Service** (SaaS) refers to cloud-based applications accessible through web browsers, desktop clients, or APIs. Users typically pay a subscription fee, and SaaS offers benefits such as automatic upgrades and data protection, as application data is stored in the cloud.
- **Platform-as-a-Service** (PaaS) is useful for software developers, providing an on-demand platform with hardware, software stack, infrastructure, and development tools. Cloud providers host everything, allowing developers to easily manage and scale applications. PaaS often involves container technology, like Docker, which virtualizes the operating system.
- **Infrastructure-as-a-Service** (IaaS) offers on-demand access to fundamental computing resources over the internet, including physical and virtual servers, networking, and storage. Users can scale resources as needed, reducing the need for large upfront investments. Unlike SaaS and PaaS, IaaS provides users with granular control over computing resources.

Another less common model is serverless computing, or **Function-as-a-Service** (FaaS). It delegates backend infrastructure tasks like provisioning and scaling to the cloud provider. This allows developers to focus only on application code and business logic. Serverless operates on a per-request basis, automatically scaling infrastructure in response to demand. Users only pay for resources used during application runtime, eliminating costs for idle capacity. FaaS allows developers to execute specific functions in response to events, with the cloud provider managing everything else—physical hardware, virtual machine operating system, and web

server software—in real-time. Billing for FaaS starts when execution begins and stops when it concludes, making it advantageous [10].

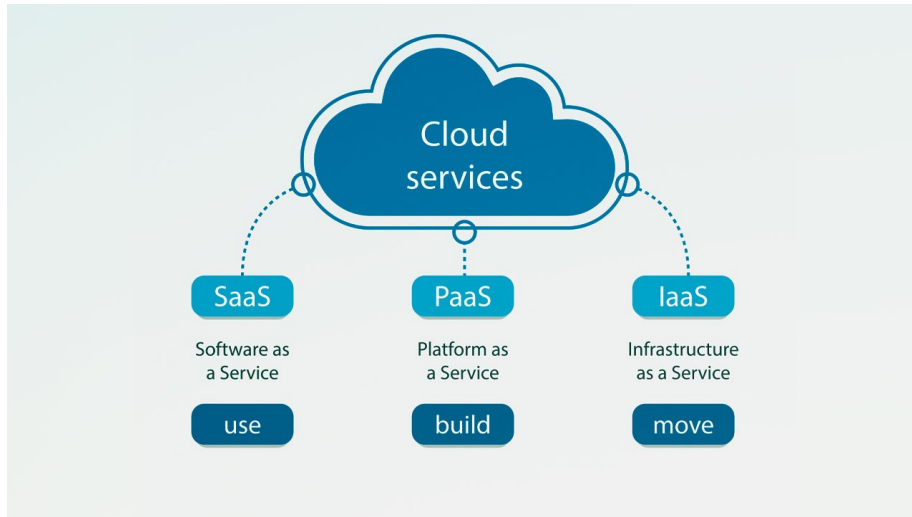


Figure 16: Three main cloud computing models [12]

Cloud computing comes in various forms, each tailored to specific needs:

1- Public Cloud

Public cloud involves a cloud service provider delivering computing resources over the public internet. These resources range from Software-as-a-Service (SaaS) applications to individual virtual machines, hardware, and complete infrastructures. Access can be free or sold via subscription or pay-per-usage models. The provider is responsible for managing data centers, hardware, and infrastructure, ensuring high-bandwidth network connectivity. Public cloud operates in a shared environment, hosting millions of customers in leading platforms like Amazon Web Services (AWS), Google Cloud, IBM Cloud and Microsoft Azure. Many enterprises are transitioning to public cloud for its scalability, elasticity, and efficiency, allowing flexible adjustments to changing workloads while minimizing resource waste and reducing spending on on-premises infrastructure.

2- Private Cloud

Private cloud is an environment where all cloud infrastructure and resources are dedicated to and accessible by a single customer. It combines cloud benefits like elasticity and scalability with the control, security, and customization of on-premises infrastructure. Typically hosted in the customer's data center, a private cloud can also be hosted by an independent provider or built on rented infrastructure in an offsite data center. Private cloud is a preferred choice for companies dealing with regulatory compliance or handling sensitive data like confidential documents, intellectual property, personally identifiable information (PII), medical records, or financial data. Building private cloud architecture according to cloud-native principles allows organizations the flexibility to migrate workloads to public cloud or run them in a hybrid cloud environment when ready.

3- Hybrid Cloud

Hybrid cloud combines public and private cloud environments, connecting an organization's private cloud services with public clouds into a flexible infrastructure for running applications and workloads. The goal is to establish a blend of public and private cloud resources, allowing organizations to choose the optimal cloud for each application or workload. Hybrid cloud provides flexibility to move workloads seamlessly between the two clouds as needed, enhancing technical and business objectives more efficiently than relying solely on public or private cloud.

4- Multicloud and Hybrid Multicloud

Multicloud involves using two or more clouds from different providers, offering flexibility and access to various services. Enterprises often leverage multiple cloud services, including SaaS, PaaS, and IaaS, from leading public cloud providers. Hybrid multicloud extends this concept, combining two or more public clouds with a private cloud environment. Organizations opt for multicloud to avoid vendor lock-in, access a diverse range of services, and promote innovation. Managing multiple clouds can be challenging due to differing tools, data transmission rates, and security protocols. Multicloud management platforms provide visibility across multiple provider clouds through a centralized dashboard, aiding development,

operations, and cybersecurity teams in monitoring and managing the environment effectively [10].

3.2 CloudVeneto platform overview

CloudVeneto operates as an OpenStack-based cloud infrastructure. Users can create Virtual Machines (VMs) tailored to specific requirements, including operating systems, software configurations, and preferred hardware specifications like processors and memory size. The platform also provides storage options, encompassing block storage (attachable volumes for virtual instances) and object storage. Beyond fundamental resources, CloudVeneto extends its services to higher-level functionalities, including management of multiple resources. Despite functioning as a unified cloud service, CloudVeneto strategically distributes its resources across two distinct data centers: Padova (hosted at INFN Padova within the University of Padova's "Dipartimento di Fisica e Astronomia") and INFN Laboratori Nazionali di Legnaro. CloudVeneto is currently built on the Yoga version of the OpenStack middleware [13].

CloudVeneto infrastructure is available to the users and collaborators of the different Departments of the University of Padova, among which Information Engineering Department (DEI) [13].

In the realm of the cloud, projects, also referred to as “tenants”, serve as distinct organizational entities. Their primary function is to effectively organize and segregate users and their associated resources. Each project is allocated specific quotas delineating resource usage and covering aspects like virtual machines, cores, memory, and storage. While a project can be personalized for a single user (termed a personal project), most common scenarios involve shared projects which accommodate multiple users, often aligning with experiments, organizations, or research groups. Users have the flexibility to participate in multiple projects simultaneously, allowing them to seamlessly switch between different endeavors. Within the CloudVeneto infrastructure, projects typically align with experiments or research groups. Notably, each project designates a project manager, usually the team leader, who assumes responsibility for managing membership requests within the project, deciding whether to accept or decline these requests [13].

Cloud instances are initially deployed on private networks, requiring access through a designated gate machine. For CloudVeneto, the specific gate host

(gate.cloudveneto.it) serves this purpose. Instances created within INFN and certain UniPD DFA projects are automatically accessible from the Local Area Networks (LANs) of INFN Padova/UniPD Physics Department, and INFN-LNL. This allows users to directly connect to VMs via SSH from desktops within these locations, eliminating the need for the gate machine. If a VM within the Cloud needs to provide a service accessible from the Internet, it can be assigned a public IP. Control over service/port accessibility is managed through security groups (to be discussed later) and firewalls on relevant VMs [13].

3.3 Available solutions for EEG management in a cloud platform

After discussing the fundamentals of cloud platforms and their infrastructures, this chapter dives into a specialized healthcare domain, exploring current solutions for managing EEG signals within the realm of cloud computing. EEG data management presents distinctive challenges, and within cloud platforms, solutions try to overcome issues related to storage, analysis, and access to signals data and patients' information.

Four case studies will be considered to analyze the market existing solutions for EEG management in cloud platforms, providing technical details about the environment and the technologies used.

3.3.1 ReportFlow: an application for EEG visualization and reporting using cloud platform

ReportFlow is a cloud-based system, developed with the aim to improve the process of reporting and delivering electroencephalograms. In a scenario of continuous development of Information Communication Technology (ICT) systems, the cloud represents a practical solution to the problems of storing and sharing a large amount of electronic health records (EHR) or other types of health data, providing several benefits to the user and organization. It allows for higher productivity compared manual exchange of data. However, some security requirements for data sharing in cloud computing systems must be guaranteed. Thus, the provider must ensure data security and privacy of sensitive information, especially in complex domains like healthcare [14].

The application has been developed to solve the problem related to the delay in reporting EEGs and evoked potentials of children at the IRCCS Centro Neurolesi Bonino Pulejo in Messina, Italy. The delay was mostly due to geographical distance and the need for a neuropsychiatrist to get to the structure each time. This caused delays up to a couple of days. The team developed, in Python programming language, a PC application called ReportFlow for sharing instrumental examinations among members of a clinical team including staff from different units. ReportFlow exploits the public cloud platform and a PKI (encryption) system for security warranty [14].

The cloud platform used in this study is Google Drive online storage of G-Suite. The provider's account (i.e., the IRCCS), which has unlimited storage space, is used to set and manage a shared drive with employees. The shared drive contains three different folders, which are accessible to employees according to their responsibilities (roles) within the company. The first folder, shared in read-only mode, includes all certificates; the second folder, shared in read/ write mode, includes the EEGs recorded (i.e., XML files containing both the EHR and the diagnostic examination, for each patient); and the third folder shared only to physicians and administrative staff, includes the EEGs reported. To keep in sync, the local folder with Cloud folder the Google Drive File Stream application was used, running on Windows and Mac OSX, while the Google Drive Ocamlfuse was used on Linux OS. Figure 17 highlights the workflow and delivery process, while Figure 18 highlights the related specific flow chart [14].

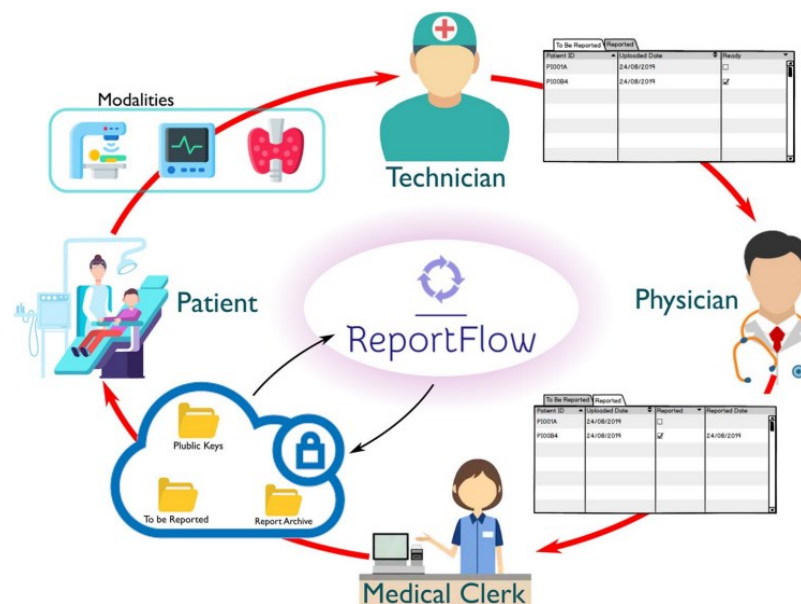


Figure 17: The EEG reporting and delivering process [14].

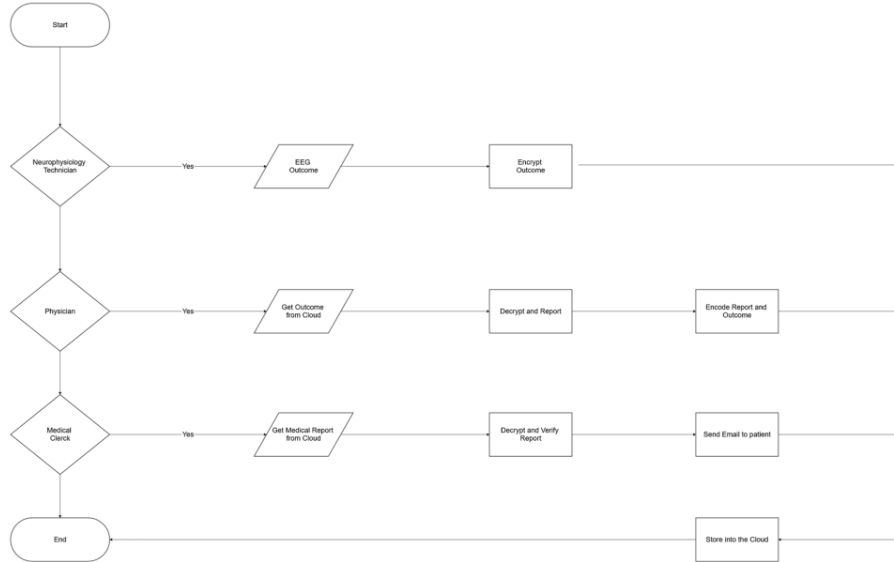


Figure 18: Flow chart of the EEG reporting and delivering process [14].

The use of ReportFlow demonstrated that there has been a significant reduction of average times in both EEG exam reporting ($t=19.94$; $p<0.001$) and delivering ($t=14.95$; $p<0.001$). Moreover, the rate of phone calls to patients was significantly lower ($\chi^2=94.87$; $p<0.001$), the number of EEG/EP exams performed increase of 20%, and the child neuropsychiatrist was able to visit about 30% of outpatients more than before. Finally, with the introduction of ReportFlow, about 68% of exam reports were delivered completely digitally [14].

Therefore, the use of ReportFlow supported the hospital in cost-saving (e.g. for paper, stationery, phone calls) and facilitated the patients as well. Using ReportFlow the reporting process becomes independent by the location: technicians can take the diagnostic examination everywhere, also in patients' homes using a portable EEG recorder, and the physician can visualize and evaluate the EEG tracing at any time, even from a remote location. Moreover, the EEG report is instantly available, and the administrative staff can archive it in real-time, while the application automatically delivers it to the patient. The comparative pre-post analysis showed promising preliminary results of performance, although the application is still in

the testing phase. Notably, the report delivering service was sensitively speeded up due to the improvement of the whole process [14].

3.3.2 EMOTIV: Mobile and Secure EEG Cloud Database

Emotiv Inc. is a privately held bio-informatics and technology company developing and manufacturing wearable electroencephalography (EEG) products including neuroheadsets, software development kits (SDK), software, mobile apps, and data products. The company uses a shared cloud platform to save EEG data. With EMOTIV Cloud, brain data collected using company's headsets and software suite is automatically and securely captured in a cloud platform. This enables unlimited storage, fast processing, and secure internal behavioral and brain data comparison without the constraints of labs or local machines. EMOTIV Cloud unlocks new use cases for researchers, developers, enterprises and individuals to securely collect, backup and analyze a trove of EEG data anywhere [15].

Users can securely store and access your EEG data from any location, knowing that it is fully protected and private. EMOTIV employs industry-standard encryption protocols to guarantee the secure transfer and storage of EEG data.

In fact, EEG datasets are automatically uploaded to the platform, allowing users to access them on multiple devices from any location and to share findings instantly with the whole team. The advanced capability in mobile EEG data recording and sharing goes beyond the constraints of traditional EEG data collection methods [15].

When group of neuroscientists, statisticians, and physicists engages in brain research, they utilize an anonymized edition of EEG data housed in EMOTIV Cloud. Through actively participating in EMOTIV EEG cloud database, developers, researchers, and citizen scientists play a vital role in expediting comprehension of the human brain. The data generated by the community amplifies EMOTIV capacity to refine algorithms and precisely gauge EEG signals for the benefit of our users.

EMOTIV Cloud is available by default in all EMOTIV applications and is supported by all available headsets. The workflow is very simple, a recording system is made of a headset (Figure 15) and a software named EmotivPRO available on App Store and Play Store.



Figure 19: EPOC-X headset by EMOTIV [15]

After data acquisition, user has the possibility to analyze and save data recordings locally or to secure them in EMOTIV cloud (Figure 20).

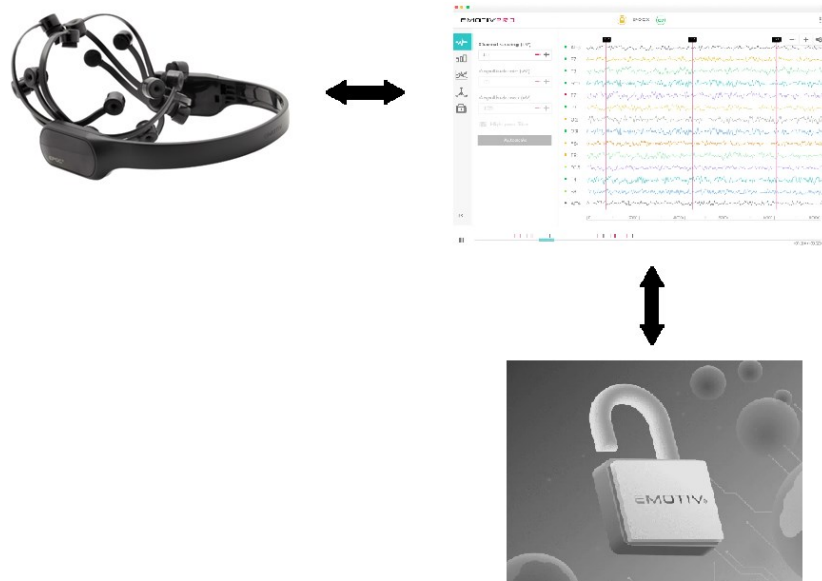


Figure 20: EmotivPRO basic workflow [15]

3.3.3 Brain Science: A Cloud-based Data Platform for Efficient EEG Data Management, Collaboration, and Analysis

To ensure the overall success of a neuroscience study, multiple researchers may collaborate concurrently to accelerate the progress of the study in different tasks. Therefore, a platform that can improve the efficiency of each task is critical for accelerating the entire research project. The collected data represent valuable

resources that warrant careful management for data security and to support subsequent analyses [16]. The availability of hardware resources, such as computing power, could also limit researchers in their choice of specific analysis strategies. While a host of tools and platforms exist to aid in various aspects of EEG research, the need for a comprehensive solution that combines data management, collaboration, and data analysis remains a key challenge.

The study introduces a cloud-based data analysis collaboration platform to address the broader needs of neuroscience research beyond EEG data analysis. Recognizing the collaborative nature of tasks like experiment arrangement and data collection, the platform aims to enhance efficiency in data management, research collaboration, and EEG data analysis. By providing cloud-based services for storage, computing, analysis, sharing, and collaboration, the platform streamlines the development of EEG-related experiments.

The platform's system architecture follows a browser-server model (Figure 21). Its backend, hosted on a cloud server, allows users to interact with cloud-based data through a front-end browser. The cloud server includes an administrative database, an EEG file database, and a computation server for storing administrative data, EEG files, and processing extensive EEG data, respectively. The platform comprises four core modules: data management, data visualization, data query, and data analysis. Users can wirelessly upload EEG data or use a hard drive, supporting common neuroscience storage formats like EDF, BDF, and CNT. Uploaded data undergoes a quality check, with accepted data stored on the file server. JavaScript frameworks facilitate data visualization, and Elasticsearch powers the search function, constructing retrieval indexes and sorting files based on signal similarity. The analysis function employs the MNE algorithm library, allowing researchers to upload and share custom algorithm files for EEG analysis.

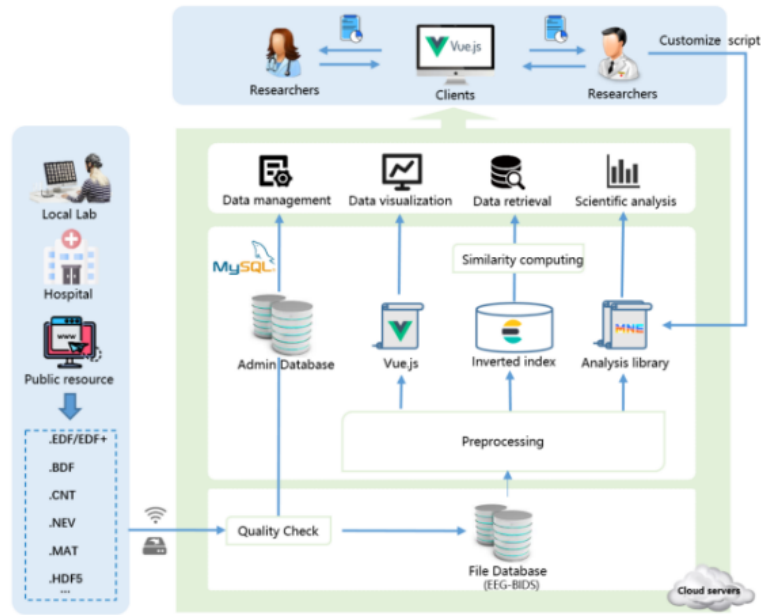


Figure 21: Architecture of the platform, starting from local laboratories to cloud data storage.

The research administrator can assemble a team, enabling collaboration and record-keeping on the platform. During the data collection phase, researchers can create projects to document essential details of EEG experiments, including subject information and experiment schedules. All experiment-related data, such as location, duration, participants, and devices used, is recorded on the platform for efficient management and retrieval. The platform also facilitates easy uploading of experiment-generated data, associating it with relevant details such as the operator, subject, and device. In the data analysis phase, researchers responsible for analysis tasks can utilize the platform's built-in support for common analysis tasks like EEG data filtering, time-frequency analysis, and EEG topomaps. They can also upload custom EEG analysis code files. The computation server executes defined analysis tasks sequentially, storing the data and results on the platform. The platform provides a user-friendly interface for analysis functions, eliminating the need for researchers to write and execute code. Researchers can save defined parameters as fixed schemes, streamlining future analyses. The platform also allows researchers to choose whether to make custom schemes public or private, facilitating collaboration among researchers [16].

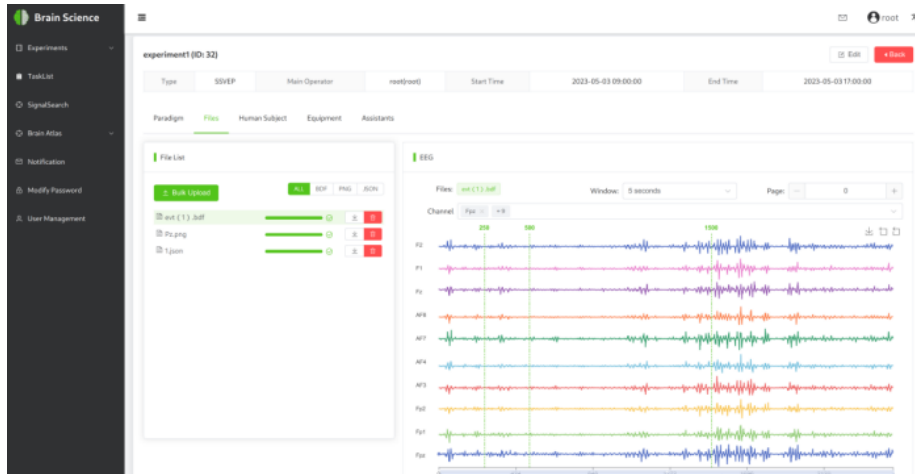


Figure 22: Brain Science platform user interface [16]

The platform facilitates the collection, management, and computation of large batches of EEG data, ultimately enhancing the efficiency of EEG experiments and promoting the development of neuroscience research [16].

3.3.4 Cloud Infrastructure for Storing and Processing EEG and ERP Experimental Data

The field of electroencephalography (EEG) and event-related potentials (ERP) involves expertise from diverse domains such as signal processing, database management, and hardware. Additionally, laboratory personnel handling experiments need comprehensive knowledge of the entire process, including data collection, storage, processing, and result presentation within a complex chain of activities. The adoption of cloud computing allows for the distribution of responsibilities, which can be either partially outsourced to a paid service or divided among individual experts who may not necessarily be situated in the same laboratory. The team developed a complex cloud platform for storing and processing experimental data. It contains a data storage, a library of signal processing methods, and a simple GUI allowing users to easily control the whole system [17].

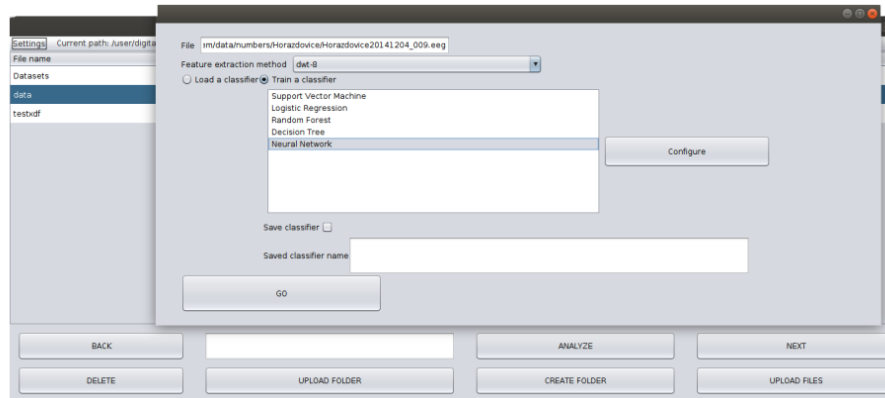


Figure 23: Graphic User Interface of the platform [17]

The platform has been validated using a deep learning analysis algorithm on ERP signals. Key aspects include robust and secure data storage for collected data, linear execution of data processing operations with the need for parallel execution in tasks like classification employing deep learning methods [17].

Commonly employed open-source technologies for data storage and processing in distributed environments encompass Apache Hadoop and Apache Spark. Apache Hadoop, designed for executing massive parallel jobs, serves as a shared space for data and computational resource access. It comprises multiple technologies within its ecosystem. The Hadoop Distributed File System (HDFS) is utilized for high-throughput access to distributed data.

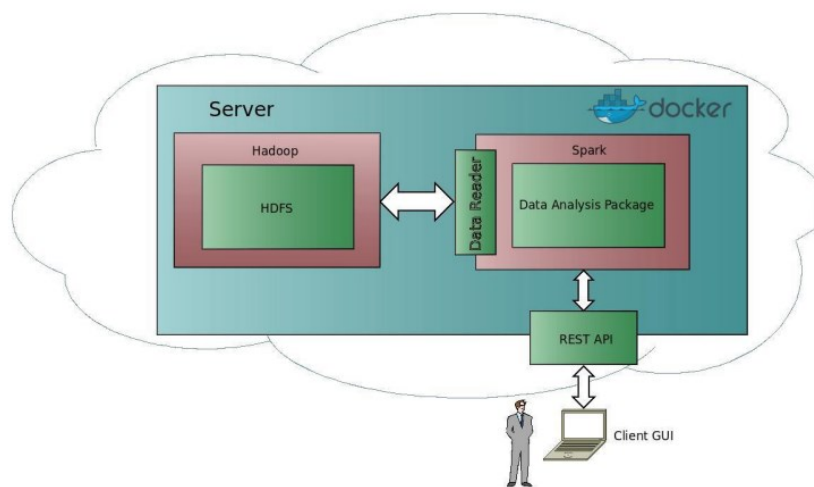


Figure 24: Platform architecture, from client to data processing on server [17]

The architecture of the system has been developed as shown in Figure 24. A cloud part of the architecture is a docker image containing on the left side: A Hadoop distributed file system (HDFS) implemented in the Cloudera platform. On the right side: The remote server operating the data analysis package. The server accesses data via a data reader and communicates with the client GUI program via the REST API.

4. Methods: platform design and implementation

This section will provide a comprehensive technical description of the plugin developed to integrate with Micromed Brain Quick Software. The entire codebase is written in Python, leveraging the Django framework for efficient management of the database, queries, and URLs.

4.1 Django core

Django is as an open-source web framework written in Python programming language. It is used by some of the largest websites in the world including Instagram, Mozilla, and NASA, but also lightweight enough to be a popular choice for weekend side projects and startups. Its "batteries-included" philosophy ensures that a skilled developer can rapidly create a robust website with a rich set of tools and features [19].

A web framework is a standardized software that simplifies and abstracts common challenges in website development. It addresses tasks such as database connection, server deployment, URL routing, security, and user registration. Python, a popular programming language, is recognized for its friendliness, power, and robust ecosystem, notably with frameworks like Django. Django, aligning with Python's philosophy, offers built-in features, robust security practices, and extensive documentation. It is known for stability, infrequent breaking changes, and a vibrant ecosystem of third-party applications. Major updates are released approximately every nine months, with regular security and bug fix patches.

Django's ecosystem includes third-party applications visible on Django Packages, with popular ones often integrated into Django itself over time. The framework is regarded for its maturity, stability, and continuous improvement within a well-defined release schedule [19].

4.2 Django components

Django deals with projects, structured and ordered folders with different files (see Figure). The main project folder, often named after the project itself, acts as the central hub. This folder contains files and subfolders, starting with the “**settings.py**” file, which contains configuration settings for the entire project, covering aspects like databases, time zones, installed apps, and static files. Management of URLs is handled by the “**urls.py**” file, specifying how different URLs should be processed and connecting them to specific views. Entry points for WSGI and ASGI servers, essential for communication between the Django application and the web server, are provided by the “**wsgi.py**” and “**asgi.py**” files. Django projects are made up of apps, each residing in its own folder. App folders contain key files like “**views.py**” for defining views, “**models.py**” for defining database models, and a “**templates**” folder for HTML templates. These apps collectively contribute to the functionality of the entire project. Database models are defined in the “**models.py**” file within each app, using Django's ORM system to structure database tables. Static files, including CSS, JavaScript, and images, are stored in the “**static**” folder. HTML templates reside in the “**templates**” folder, often organized within each app. These templates dynamically generate HTML content based on data from views, contributing to the project's overall structure. Database schema changes are tracked in the “**migrations**” folder, where developers create and apply migrations to keep the database structure updated. The “**manage.py**” file is a command-line utility offering various functionalities, such as running the development server and applying migrations, simplifying project management tasks [18].

This well-organized structure in Django facilitates the development process, making it modular, scalable, and more straightforward to manage.

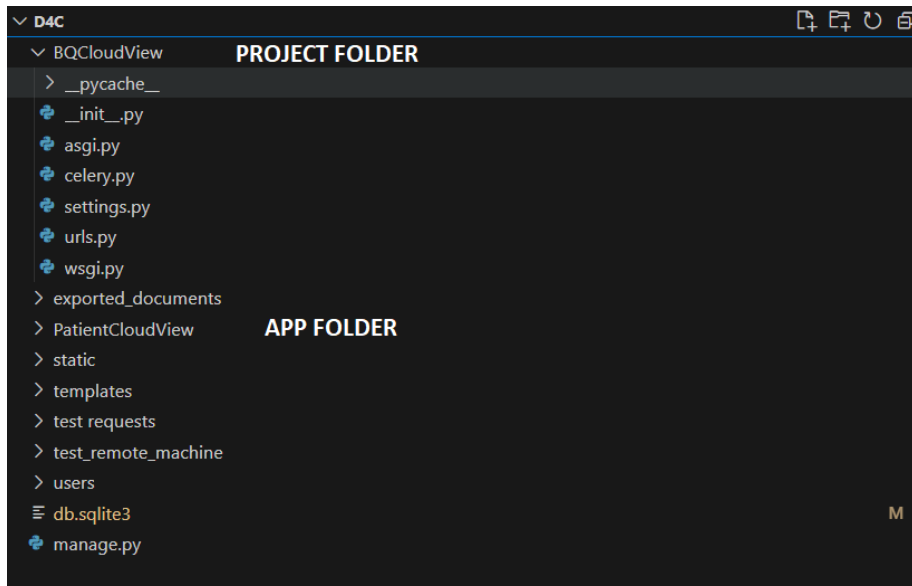


Figure 25: Example of folders structure in a Django project, with reference to project folder and app folder

4.2.1 Templates

In web development, Django relies on templates to dynamically generate HTML content. A template incorporates static HTML components and utilizes a special syntax to outline the insertion of dynamic content. Regardless of the chosen backend, Django maintains a standardized API for loading and rendering templates. Loading involves locating the template based on a given identifier and preprocessing it, typically involving compilation into an in-memory representation. Rendering encompasses the interpolation of the template with context data, ultimately producing the final string [18].

The Django template language employs a text document or a Python string as a template, incorporating specific constructs that the template engine recognizes and interprets. The fundamental elements within this language are variables and tags.

When rendering a template, it is accompanied by a context. During rendering, variables are substituted with their corresponding values, retrieved from the context, and tags are executed. The remainder of the content is output as it appears [18]. Django templates utilize a double curly brace notation, like `{{ variable }}`, for **variables**. These variables represent dynamic content to be substituted during rendering. **Tags** in Django templates are enclosed in curly braces

with percent signs, such as `{% tag %}`. Tags execute specific logic or control flow within the template. **Filters**, another construct, modify the appearance or behavior of variables in the template. They are applied using the pipe symbol, e.g., `{{ variable|filter }}` [18].

Django templates support loops and conditional constructs, facilitating dynamic content generation based on the provided context (see Figure).

```

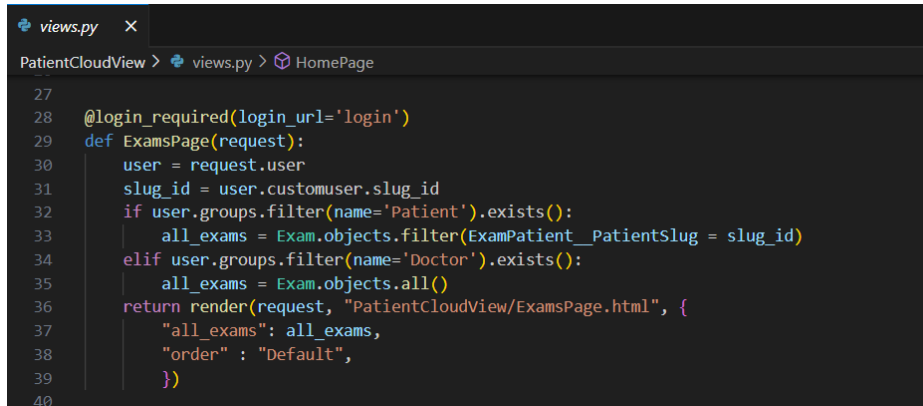
PatientCloudView > templates > PatientCloudView > HomePage.html
1  {% extends "base.html" %}
2  {% load static %}
3
4  {% block title %}
5  BQ Cloud View
6  {% endblock %}
7
8  {% block css_files %}
9  <link rel="stylesheet" href="{% static 'PatientCloudView/styles.css' %}" />
10 <link rel="stylesheet" href="{% static 'PatientCloudView/exam.css' %}" />
11 {% endblock %}
12
13 {% block content %}
14
15 <section id="welcome">
16 <header>
17 
18 <h2>Brain Quick Cloud View</h2>
19 </header>
20 </section>
21
22 <section id="latest-posts">
23
24     {% if request.user.is_authenticated %}
25     <h2>Latest exams sent</h2>
26     <ul>
27         {% for exam in latest_exams %}
28             {% include "PatientCloudView/includes/exam.html" %}
29         {% endfor %}
30     </ul>
31     {% else %}
32     <h2>Login to see the latest exams</h2>
33     {% endif %}
34 </section>
35 {% endblock content %}
36
37
38
39
```

Figure 26: Template example from Brain Quick Cloud View

4.2.2 Views

A view function, commonly referred to as a view, is a Python function designed to handle a web request and produce a corresponding web response. This response may encompass various formats, such as the HTML content of a webpage, a redirect, a 404 error, an XML document, an image, a template, or virtually any other type of content. The view function encapsulates the specific logic necessary to

generate and return the desired response. The flexibility is such that this code can reside anywhere in project folder, without any intricate requirements. Conventionally, for organizational purposes, views are often placed in a file named **views.py** within the project or application directory, but this is more of a convention than a strict rule [18].



```
views.py x
PatientCloudView > views.py > HomePage
27
28 @login_required(login_url='login')
29 def ExamsPage(request):
30     user = request.user
31     slug_id = user.customuser.slug_id
32     if user.groups.filter(name='Patient').exists():
33         all_exams = Exam.objects.filter(ExamPatient_PatientSlug = slug_id)
34     elif user.groups.filter(name='Doctor').exists():
35         all_exams = Exam.objects.all()
36     return render(request, "PatientCloudView/ExamsPage.html", {
37         "all_exams": all_exams,
38         "order" : "Default",
39     })
40
```

Figure 27: View example from Brain Quick Cloud View. This view queries all available exams, after verifying user login and group membership

4.2.3 URLs

To establish URLs for an application, it's necessary to create a Python module referred to as a URLconf (URL configuration). This module, comprised solely of Python code, functions as a mapping tool, associating URL path expressions with specific Python functions, usually representing views. Mapping can be even extensive, with the flexibility to reference other mappings [18]. URLs, conventionally, are all grouped in “**urls.py**” file.



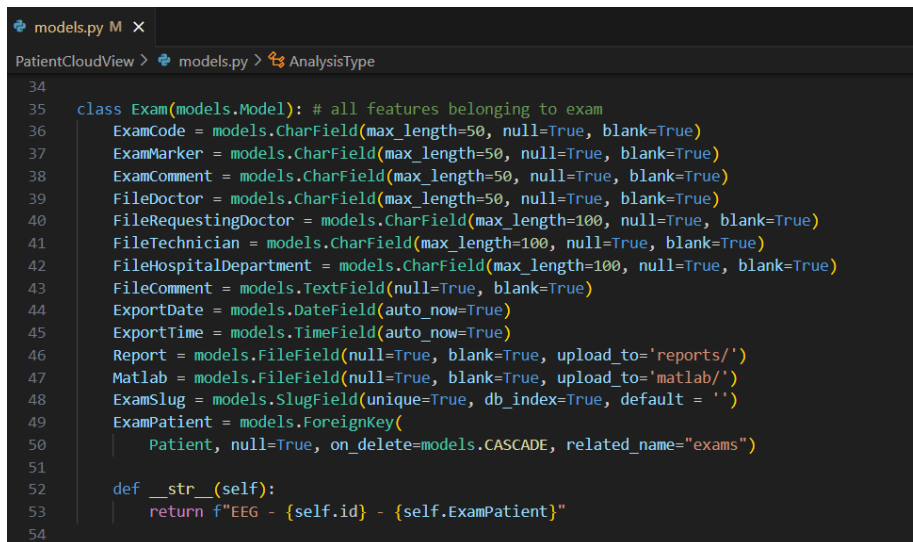
```
urls.py M x
PatientCloudView > urls.py > ...
1 from django.urls import path
2 from django.contrib.auth import views as auth_views
3 from . import views
4
5 urlpatterns = [
6     path("patients/info/<slug:PatientSlug>", views.PatientInformation, name='patient-info-page')
7 ]
8
9
```

Figure 28: URL pattern example from Brain Quick Cloud View. The dynamic URL loads PatientInformation view based on the parameter PatientSlug passed from URL.

4.2.4 Models

A model serves as the primary and comprehensive repository of information for data in Django. It includes the fundamental fields and functionalities related to the data intended to be stored, aligning with a specific database table. Each model is implemented as a Python class, inheriting from “django.db.models.Model”. Every attribute within the model corresponds to a distinct database field, defining the structure of database table.

Through this model definition, Django provides an API for seamless interaction with the database, allowing to perform queries effortlessly [18]. Each field in a model is an instance of the appropriate Field class. the strength of relational databases lies in establishing connections between tables. Django provides effective means to articulate the three predominant types of database relationships: many-to-one, many-to-many, and one-to-one [18]. Conventionally, models are saved in “models.py” document.



```
models.py M X
PatientCloudView > models.py > AnalysisType
34
35 class Exam(models.Model): # all features belonging to exam
36     ExamCode = models.CharField(max_length=50, null=True, blank=True)
37     ExamMarker = models.CharField(max_length=50, null=True, blank=True)
38     ExamComment = models.CharField(max_length=50, null=True, blank=True)
39     FileDoctor = models.CharField(max_length=50, null=True, blank=True)
40     FileRequestingDoctor = models.CharField(max_length=100, null=True, blank=True)
41     FileTechnician = models.CharField(max_length=100, null=True, blank=True)
42     FileHospitalDepartment = models.CharField(max_length=100, null=True, blank=True)
43     FileComment = models.TextField(null=True, blank=True)
44     ExportDate = models.DateField(auto_now=True)
45     ExportTime = models.TimeField(auto_now=True)
46     Report = models.FileField(null=True, blank=True, upload_to='reports/')
47     Matlab = models.FileField(null=True, blank=True, upload_to='matlab/')
48     ExamSlug = models.SlugField(unique=True, db_index=True, default='')
49     ExamPatient = models.ForeignKey(
50         Patient, null=True, on_delete=models.CASCADE, related_name="exams")
51
52     def __str__(self):
53         return f"EEG - {self.id} - {self.ExamPatient}"
54
```

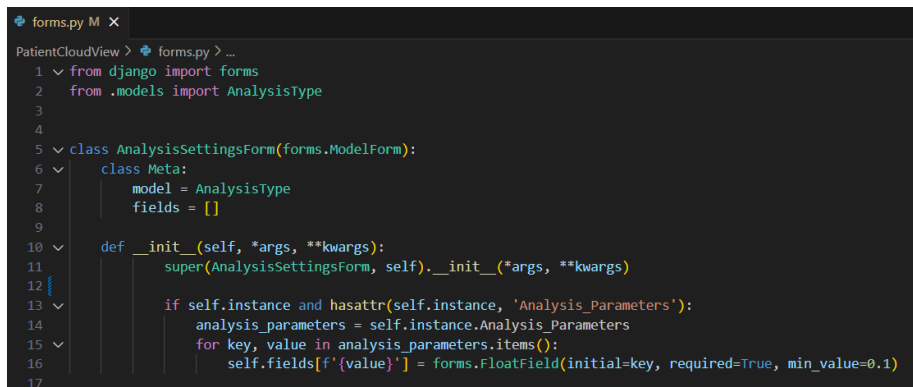
Figure 29: Example of Exam model in Brain Quick Cloud View project

Figure 29 shows how the database table has been created for the object of type “Exam”, inherited from built-in “models” class. Each field is a specific feature of the object Exam. Some are defined as Char fields, while other are Date/Time or File fields. The foreign key “ExamPatient” is a link to “Patient” model, establishing a relationship one-to-many from “Patient” to “Exam”.

4.2.5 Forms

In HTML, a form is a set of elements enclosed within `<form>...</form>` that enables users to input text, select options, and interact with controls, subsequently sending this information back to the server. The form utilizes various elements, some inherent in HTML, while others, like date pickers or sliders, require JavaScript and CSS in addition. Key attributes for a form include specifying the URL (where the data should be returned) and the HTTP method for the return (GET or POST) [18].

Django simplifies the process of managing forms within web applications. The built-in Form class maps form fields to HTML form elements, offering a streamlined solution for rendering and processing forms. Fields, represented as classes, manage data and perform validation upon submission. HTML "widgets" visualize form fields in the browser. Rendering a form in Django involves obtaining it in the view, passing it to the template context, and expanding it into HTML markup using template variables. Forms can be instantiated empty or prepopulated with data, such as information from saved model instances or previous form submissions. Django's form functionality automates these tasks securely [18].

A screenshot of a code editor window titled 'forms.py M X'. The code is written in Python and defines a Django form class. The code is as follows:

```
1 from django import forms
2 from .models import AnalysisType
3
4
5 class AnalysisSettingsForm(forms.ModelForm):
6     class Meta:
7         model = AnalysisType
8         fields = []
9
10    def __init__(self, *args, **kwargs):
11        super(AnalysisSettingsForm, self).__init__(*args, **kwargs)
12
13
14        if self.instance and hasattr(self.instance, 'Analysis_Parameters'):
15            analysis_parameters = self.instance.Analysis_Parameters
16            for key, value in analysis_parameters.items():
17                self.fields[f'{value}'] = forms.FloatField(initial=key, required=True, min_value=0.1)
```

Figure 30: Example of a form from Brain Quick Cloud View written in file `forms.py`.

In the Brain Quick Cloud View project, the creation of a form is illustrated in Figure 30. The "**forms.py**" file conventionally houses all forms intended for submission in HTML pages. Specifically, the "AnalysisSettingsForm" has been developed by extending Django's built-in Form class, utilizing the "ModelForm" attribute. This attribute facilitates the creation of a form that encompasses all fields of the specified model, exemplified by "AnalysisType" in this case. The following code snippet is designed to showcase the stored values when the form is loaded.

```
views.py x
PatientCloudView > views.py > AnalysisSettingsInput
326
327 @login_required(login_url='login')
328 @user_in_group('Doctor')
329 def AnalysisSettingsInput(request, ExamSlug, Analysis_Name):
330     identified_exam = Exam.objects.get(ExamSlug=ExamSlug)
331     identified_analysis = AnalysisType.objects.get(Analysis_Name=Analysis_Name)
332     if request.method == "POST":
333         form = AnalysisSettingsForm(request.POST, instance=identified_analysis)
334         if form.is_valid():
335             analysis_parameters = {}
336             for key in form.cleaned_data:
337                 value = form.cleaned_data[key]
338                 analysis_parameters[value] = key
339             form.instance.Analysis_Parameters = analysis_parameters
340             form.save()
341             url_reverse = reverse(AnalysisSettings, args=[ExamSlug])
342             return HttpResponseRedirect(url_reverse)
343     else:
344         form = AnalysisSettingsForm(instance=identified_analysis)
345     return render(request, "PatientCloudView/AnalysisSettingsForm.html", {
346         "identified_exam" : identified_exam,
347         "analysis" : Analysis_Name,
348         "form" : form,
349     })
350
```

Figure 31: How the form is rendered from a specific view in Brain Quick Cloud View project

Figure 31 instead shows how the form logic and how it is rendered from “AnalysisSettingsInput” view. In particular, the form containing analysis parameters is rendered with the actual parameters saved and, if the request method is POST, new values submitted by the user are saved and user is redirected to another web page.

Eventually, Figure 31 shows how the form is managed from HTML code in “AnalysisSettingsForm” template.

```
AnalysisSettingsForm.html M X
PatientCloudView > templates > PatientCloudView > AnalysisSettingsForm.html
24
25     {% if request.user.is_authenticated %}
26     <a href="{% url 'analysis-settings' identified_exam.ExamSlug %}" > 
28         <div class="form-control">
29             <p><strong>Modify {{analysis}} cutoff frequency (Hz)</strong></p>
30             <form method="post">
31                 {% csrf_token %}
32                 {% for field in form %}
33                     {{ field.label }}
34                     {{ field }}
35                     {{ field.errors}}
36                     <br>
37                 {% endfor %}
38                 <button type="submit">Modify</button>
39             </form>
40         </div>
41     </article>
42
43
44
45
```

Figure 32: Definition of the form in HTML page

Form and its method are defined using `<form method = "post">` and button for submission.

4.2.6 REST Framework

Django REST Framework (DRF) is a versatile toolkit within the Django ecosystem, designed specifically for constructing Web APIs. It enhances the capabilities of the Django web framework, providing a set of tools to create, version, and use APIs effectively. DRF encompasses essential features such as serialization, authentication, permissions, viewsets, and routers, allowing developers to establish robust and scalable RESTful APIs seamlessly. Its aim is to simplify common tasks associated with API development, while also encouraging adherence to best practices, fostering code reuse, and promoting long-term maintainability [20]. Conventionally, API folder is created inside application folder, and it includes **“urls.py”**, **“views.py”** (specific for APIs management) and **“serializers.py”**.

File containing URLs specifies which are the links that serve for HTTP requests, while file containing Views indicates how views, associated singularly to URLs manage different types of requests (GET, POST, PUT, DELETE). Eventually, **“serializers.py”** serves as a tool for request serialization, useful to manage different type of requests.

```
serializers.py x  urls.py  views.py
PatientCloudView > api > serializers.py > ...
15 class ExamSerializer(serializers.ModelSerializer):
16     class Meta:
17         model = Exam
18         fields = "__all__"
19
20     def create(self, validated_data):
21
22         exam = Exam.objects.create(**validated_data)
23         return exam
24
25     def update(self, instance, validated_data):
26         instance.ExamCode = validated_data.get('ExamCode', instance.ExamCode)
27         instance.ExamMarker = validated_data.get('ExamMarker', instance.ExamMarker)
28         instance.ExamComment = validated_data.get('ExamComment', instance.ExamComment)
29         instance.FileDoctor = validated_data.get('FileDoctor', instance.FileDoctor)
30         instance.FileRequestingDoctor = validated_data.get('FileRequestingDoctor', instance.FileRequestingDoctor)
31         instance.FileTechnician = validated_data.get('FileTechnician', instance.FileTechnician)
32         instance.FileHospitalDepartment = validated_data.get('FileHospitalDepartment', instance.FileHospitalDepartment)
33         instance.FileComment = validated_data.get('FileComment', instance.FileComment)
34         instance.Report = validated_data.get('Report', instance.Report)
35         instance.Matlab = validated_data.get('Matlab', instance.Matlab)
36
37         instance.save()
38         return instance
```

Figure 33: Example of serializer to manage creation and update request for Exam object from Brain Quick Cloud View

Figure 33 shows how “ExamSerializer” is defined, to manage creation and update requests to the exam endpoint.

4.3 Brain quick cloud architecture

After discussing basic some basic Django concepts used for this project, it’s now time to dive deeply into platform architecture and details. Django project is named BQCloudView, with PatientCloudView being the only associated application, containing all features that will be discussed in the next chapters.

The application is composed of a set of views and related URLs to dive into patients and exams data received from Micromed File Manager. The objective of the project is to be installed in a cloud platform to easily visualize patients and exams data, download reports generated from hospital technicians and allow EEG data analysis for clinical purposes.

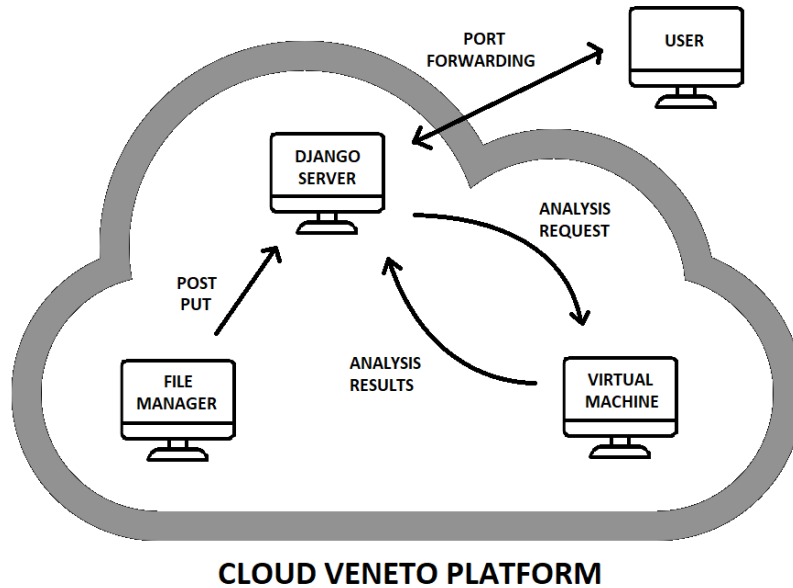


Figure 34: Basic platform description

Figure 34 shows the basic architecture of the platform, in particular:

- 1- **File Manager machine** installed in CloudVeneto platform contains all clinical data that clinicians need to export to Django server. Data exported from File Manager are sent to Django server endpoints, that manage:
 - a. New **Patient insertion** with all data available from File Manager (POST request)
 - b. **Patient data update** with all data available from File Manager (PUT request)
 - c. New **Exam insertion** with all data available from File Manager, together with exam Report and exam Data in Matlab format (POST request)
 - d. **Exam data update** with all data available from File Manager, together with exam Report and exam Data in Matlab format (PUT request)

- 2- **Django server machine** installed in CloudVeneto platform is equipped with the entire project code essential for initiating the server, ensuring its continuous operation, and maintaining a stable workflow.

- 3- Every time a user initiates an EEG analysis through the Django server, a **Virtual Machine** is dynamically created in the cloud platform. This virtual machine processes the incoming request along with the associated data, performs the analysis, and subsequently transmits the results back to the Django server. This setup ensures that the data analysis is seamlessly handled by a separate machine, enhancing efficiency in the process.
- 4- **End User** can easily access data from Django server thanks to CloudVeneto port forwarding feature.

4.3.1 Models definition

For this project purpose it's has been decided to use patients and exams information managed from File Manager, to make the integration easier to implement, replicating the same fundamental fields. In addition, other models have been defined to work with data analysis and to manage user permissions.

Patient model

```
class Patient(models.Model): # all features belonging to
patient
    ID1 = models.CharField(max_length=50, null=True,
blank=True)
    ID2 = models.CharField(max_length=50, null=True,
blank=True)
    LastName = models.CharField(max_length=50, null=True,
blank=True)
    FirstName = models.CharField(max_length=50, null=True,
blank=True)
    Birthdate = models.CharField(max_length=50, null=True,
blank=True)
    Gender = models.CharField(max_length=50, null=True,
blank=True)
```

```
    Height = models.CharField(max_length=50, null=True,
blank=True)
    Weight = models.CharField(max_length=50, null=True,
blank=True)
    Address = models.CharField(max_length=50, null=True,
blank=True)
    City = models.CharField(max_length=50, null=True,
blank=True)
    Country = models.CharField(max_length=50, null=True,
blank=True)
    HospitalDepartment = models.CharField(max_length=50,
null=True, blank=True)
    Marker = models.CharField(max_length=50, null=True,
blank=True)
    Doctor = models.CharField(max_length=50, null=True,
blank=True)
    PatientSlug = models.SlugField(unique=True,
db_index=True, default = '')
    Comment = models.TextField(null=True, blank=True)
```

Patient model contains the following fields:

- **ID1**, equivalent to patient univocal code
- **ID2**, equivalent to tax code
- **LastName**
- **FirstName**
- **Birthdate**
- **Gender**
- **Height**
- **Weight**
- **Address**
- **City**
- **Country**
- **HospitalDepartment**

- **Doctor**
- **PatientSlug**, built as FirstName-LastName-Birthdate, it is the patient identifier in Django Server
- **Comment**

Exam model

```
class Exam(models.Model): # all features belonging to exam
    ExamCode = models.CharField(max_length=50, null=True,
blank=True)
    ExamMarker = models.CharField(max_length=50, null=True,
blank=True)
    ExamComment = models.TextField(null=True, blank=True)
    ExportDate = models.DateField(auto_now=True)
    ExportTime = models.TimeField(auto_now=True)
    Report = models.FileField(null=True, blank=True,
upload_to='reports/')
    Matlab = models.FileField(null=True, blank=True,
upload_to='matlab/')
    ExamSlug = models.SlugField(unique=True, db_index=True,
default = '')
    ExamPatient = models.ForeignKey(
        Patient, null=True, on_delete=models.CASCADE,
related_name="exams")
```

Exam model contains the following fields:

- **ExamCode**, equivalent to exam univocal code
- **ExamComment**
- **ExportDate**, it's the date related to data sending to Django server
- **ExportTime**, it's the time related to data sending to Django server
- **Report**, it's a file type field, used to store report exported from File Manager

- **Matlab**, it's a file type field, used to store Matlab EEG data exported from File Manager
- **ExamSlug**, built from Exam ID in File Manager database, it is the exam univocal identifier in Django Server
- **ExamPatient**, it's a foreign key field which serves as a link with Patient model, the relationship is one-to-many

Analysis Type model

AnalysisType model contains the following field, and serves to keep trace of all available analysis from Django Server:

- **Analysis_Name**
- **Analysis_Parameters**, it's a JSON type field, used to store all information needed for analysis

Analysis model

```
class Analysis(models.Model): # all features belonging to
exam
    Analysis_Report = models.FileField(null=True,
blank=True, upload_to='report_analysis/')
    Analysis_Matlab = models.FileField(null=True,
blank=True, upload_to='matlab_analysis/')
    AnalysisExportDate = models.DateField(auto_now=True)
    AnalysisExportTime = models.TimeField(auto_now=True)
    AnalysisSlug = models.SlugField(db_index=True, default =
'')
    AnalysisExam = models.ForeignKey(
        Exam, null=True, on_delete=models.CASCADE,
related_name="analysis")
    Analysis_Type = models.ForeignKey(
        AnalysisType, on_delete=models.CASCADE,
related_name="analysis_type")
```

Analysis model contains the following fields:

- **Analysis_Report**, it is the report generated from the execution of the analysis
- **Analysis_Matlab**, it's the Matlab file generated from the execution of the analysis (cleaned data)
- **AnalysisExportDate**, it's the date related to data sending back to Django server
- **AnalysisExportTime**, it's the time related to data sending back to Django server
- **AnalysisSlug**, it's the identifier of the analysis executed
- **AnalysisExam**, it's a foreign key field which serves as a link with Exam model, the relationship is one-to-many
- **Analysis_Type**, it's a foreign key field which serves as a link with AnalysisType model, the relationship is one-to-many

Analysis Pipeline model

AnalysisPipeline model contains the following fields, and serves to manage a sequence of maximum three analysis in a row:

- **First_Analysis**
- **Second_Analysis**
- **Third_Analysis**

Custom User model

CustomUser model contains the following fields, and it is used to manage user permissions:

- **User**, it's a one-to-one field that extends the default User model
- **Slug_id**, it's the ID used from Django Server to match logged in user with existing patients

4.3.2 Users management

Django Server is capable of managing Users permissions based on the membership group. In particular, users belonging to **Doctor** group have the full control over functionalities in Brain Quick Cloud View, including:

- View of all available patients and exams
- Patient deletion from server
- Exam deletion from server
- View of all available analysis
- Analysis creation request
- Analysis deletion from server
- Matlab data and Report data download

Users belonging to **Patient** group instead can only access to their own patient and exams data. They cannot create analysis requests and cannot delete records from database.

Once a new user is created in Django Server database, it is necessary to choose the belonging group and to fill in Slug ID field, which is used to match the patient with data exported from File Manager. See how to create and modify users in Useful Documents section.

4.3.3 Analysis management

The major functionality of this project revolves around the capability to initiate an EEG analysis based on an available exam on Django server. When a user selects an exam, he can navigate to the "Go to Analysis" section, where analysis parameters can be configured, and an analysis algorithm can be applied to EEG data. Users can easily execute an analysis by choosing it from the interface. The project integrates three basic predefined analysis types:

- Mean
- Notch filtering
- Bandpass filtering

A notable aspect is the extensibility of the project, allowing the addition of new analysis algorithms with their specific parameters to the Django database. This flexibility makes the project more versatile and useful.

It's important to highlight that signal processing is not carried out on the Django server itself but is delegated to other workers within the same Server network. These workers are configured with a Python environment and a script that continuously listens on a configurable port, facilitating communication between the server and workers. When a user requests an analysis, a POST request is dispatched to the worker machine, including details such as the analysis type, parameters, and EEG data in Matlab format. The script on the worker machine interprets the data, converts it into a Python-readable format, and initiates the analysis, generating a new Matlab file with the analyzed data and a Report file containing key analysis points (e.g., name and parameters). Upon completion of the analysis, the worker sends a POST request back to Django server APIs, which handle the insertion of the new analysis data. In the context of this project, where the web application operates on the CloudVeneto platform, worker machines are dynamically created whenever a new analysis request is initiated. This approach contributes to efficient utilization of computational resources. The overall analysis workflow is illustrated in the following image:

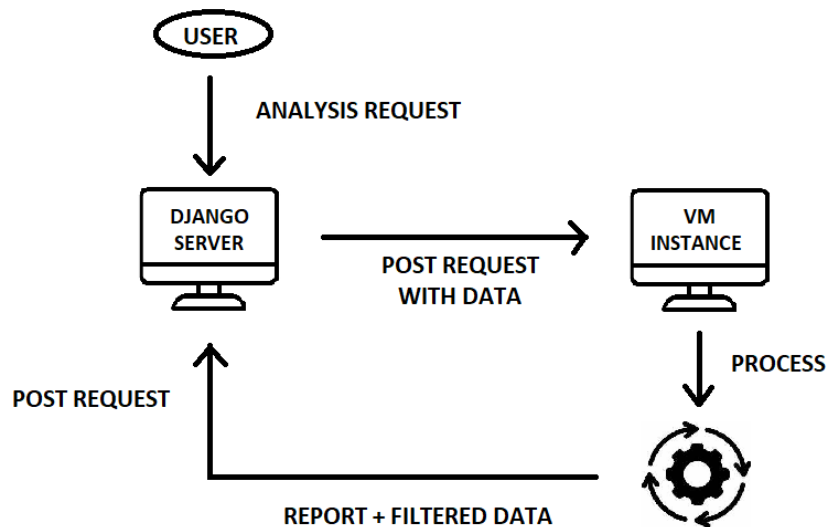


Figure 35: Basic workflow of a new analysis request from Django server

Employing the identical workflow, a user has the capability to execute multiple analyses consecutively, organized within a structure referred to as a "Pipeline." Specifically, users can select from existing analysis types to construct and store their personalized pipeline or initiate a new one, with up to three analyses in a row to be executed on data. The analysis process remains consistent, involving the instantiation of a worker machine as soon as another script is in progress. Communication is managed through a web socket, maintaining the same characteristics as described previously. Analysis workflow will be overviewed better in the next chapters.

4.3.4 Celery Queue Manager

Celery is a simple, flexible, and reliable distributed system to process vast amounts of messages, while providing operations with the tools required to maintain such a system. It's a task queue with focus on real-time processing, while also supporting task scheduling [21].

Task queues serve as a mechanism for distributing tasks across threads or machines, with each task representing a unit of work. Dedicated worker processes continually monitor these queues for new tasks. Celery, a task queue system, relies on messages and employs a broker to mediate between clients and workers. When a client initiates a task, a message is added to the queue, and the broker delivers it to an available worker [21]. In Brain Quick Cloud View conditions, Django Server serves as a broker client, while the instantiated Virtual Machine serves as a worker that receive the message and starts processing.

Celery operates through messages, offering flexibility in language choice. While primarily written in Python, it supports other languages like Node.js and PHP through node-celery and a PHP client, respectively. This flexibility allows for language interoperability, enabling the exposure of an HTTP endpoint and utilizing tasks that request it, known as webhooks. This approach ensures compatibility and communication across different language environments. Celery relies on a message transport system for message exchange: in this project RabbitMQ was used as a broker to collect and spread messages through workers.

For this purpose, RabbitMQ needs to be installed in Django Server machine and needs to be started with the following command starting from Application folder:

celery -A BQCloudView worker

To integrate Celery queues management, it was necessary to create the file “**celery.py**” inside project folder, with the following code architecture:

```
from __future__ import absolute_import, unicode_literals
import os
from celery import Celery

# Set the Django default settings module for the 'celery'
program
os.environ.setdefault('DJANGO_SETTINGS_MODULE',
    'BQCloudView.settings')

# create an instance of celery to manage project queues
app = Celery('BQCloudView')

# load task modules from all registered Django app
configurations
app.config_from_object('django.conf:settings',
    namespace='CELERY')

# discover automatically tasks to be executed
app.autodiscover_tasks()
```

It was then necessary to create a list of tasks that will be recalled from Django views to execute analysis and pipelines. All tasks have been defined in module “**tasks.py**” in Application folder (see example below).

```
from __future__ import absolute_import, unicode_literals
from celery import shared_task
from .utils import execute_analysis, execute_pipeline,
execute_workflow
import logging
```

```

logger = logging.getLogger(__name__)

@shared_task
def analysis_execution_task(PatientSlug, ExamSlug,
Analysis_Name, Matlab_Path, Matlab_Name,
Analysis_Parameters):
    try:
        print("Execution of analysis_execution_task...")
        execute_analysis(PatientSlug, ExamSlug,
Analysis_Name, Matlab_Path, Matlab_Name,
Analysis_Parameters)

        print("analysis_execution_task completed")
    except Exception as e:
        logger.error(f"Error executing
analysis_execution_task: {str(e)}")
        raise e

```

In this code snippet, “analysis_execution_task” is defined to process the analysis with the parameters as arguments. The task is invoked directly from “AnalysisExecution” view located in module “**views.py**”. Once the user, working from Django Server, selects an exam and sends an analysis request, message is added to the queue with RabbitMQ broker and then processed, instantiating a Virtual Machine that will serve as worker to execute the analysis and send back the results (see Figure 35). If the RabbitMQ worker is correctly running, workflow is the following:

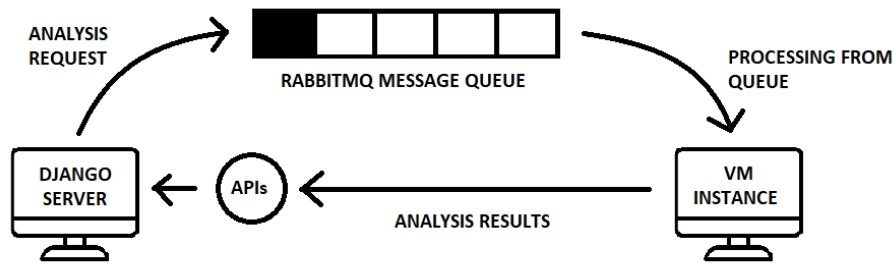


Figure 36: Analysis and Pipeline execution workflow with Celery integration and RabbitMQ message broker

4.3.5 Integration with File Manager

The integration with File Manager relies on the utilization of endpoints offered by File Manager Core, a system component responsible for communication and tools management. These endpoints provide details regarding resources, patients, exams, and associated files. Specifically designed for transitioning File Manager into a web application in future, these endpoints have undergone thorough exploration and testing within the scope of this project. The key endpoints made available by File Manager are the following:

Patient	
GET	/api/Patient/resources
GET	/api/Patient
GET	/api/Patient/exams
GET	/api/Patient/examFiles
GET	/api/Patient/examFileContent

In this way, Django database that stores all data is synchronized with File Manager database. Using Celery Beat plugin, a tool that allows to execute tasks periodically, a new task has been created to manage periodic communication with File Manager (by default 30 minutes) and keep the database synchronized. “**celery.py**” has been enriched with the following code snippet to manage Celery Beat integration:


```

app.conf.beat_schedule = {
    'exams_discovery_task': {
        'task':
'PatientCloudView.tasks.exams_discovery_task',
        'schedule': 1800, # 1800 seconds (30 minutes)
    },
}

```

Beat can be started with the following command line, and subsequently all tasks defined in Beat configuration will be executed periodically by the worker:

celery -A BQCloudView beat

The architecture ideated for “exams_discovery_task” is the following:

- 1 A GET request is sent to File Manager Core resource endpoint to retrieve all available resources in the configuration:

```

response_resources = requests.get("http://" + host +
":50050/api/Patient/resources")

```

- 2 For every resource, even though there is usually just one common resource, a GET request is sent at the patients' endpoint to gather details about all the patients currently available. Subsequently, a dictionary is instantiated based on the File Manager's response to organize data transmission to Django endpoints. In the initial step, a PUT request is dispatched to Django endpoints to verify the existence of the patient and update the information if necessary. If the request encounters an issue (404 response), a POST request is triggered to save the patient's information in the Django database.

```

for resource in resources_list:
guid = resource['guid'] #look for all available patients

```

```
response_patients = requests.get("http://" + host +
":50050/api/Patient?resourceId=" + guid +
"&paginationObject=%7B%22skip%22%3A0%2C%22take%22%3A50%7D")
```

- 3 The exams endpoints are the next focus, involving the dispatch of a GET request to File Manager Core to retrieve details about all existing exams for each patient. Even in this case, a dictionary is created to organize data sending to Django endpoints. The process proceeds exclusively when the exam type is identified as "EEG trace." Subsequently, the task advances by querying the files endpoint to obtain both the EEG trace file and the Report file, saving them locally. The EEG trace file is further transformed into a Matlab file using the Mat2Trc Plugin.

```
#look for all available exams
response_exams = requests.get("http://" + host +
":50050/api/Patient/exams?resourceId=" + guid +
"&patientId=" + str(patient['id']))
```

```
#look for all available files
response_files = requests.get("http://" + host +
":50050/api/Patient/examFiles?resourceId=" + guid +
"&patientId=" + str(patient['id']) + "&examId=" +
str(exam['id']))
```

- 4 Once the details of the exam and its associated Matlab and Report files are successfully obtained, the corresponding Django Server endpoints are accessed, and the exam with its related files is added into the Django database. Even in this case, a PUT request is sent to check if the exam is already existing. If the response is 404, the POST request is sent to create the exam.

Figure 36 shows a scheme to understand better how communication is organized to keep synchronized File Manager databases and Django server database.

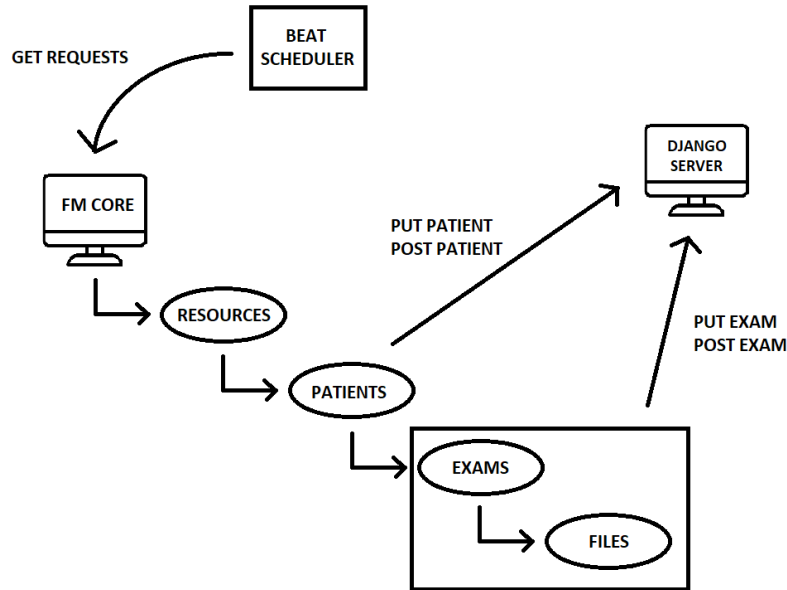


Figure 37: Integration with File Manager scheme

4.4 Front-end functionality

Following the exploration of the core of this project's architecture, let's look at the User Interface of the application. To enhance comprehension of certain technical aspects regarding the architecture, examples will be explained within the context of a single machine (localhost). However, it's important to note that these concepts can extend to scenarios where tasks are executed across diverse machines, as will be explained in the installation process on the CloudVeneto platform. The comprehensive explanation of the project's functionalities will be organized as follows:

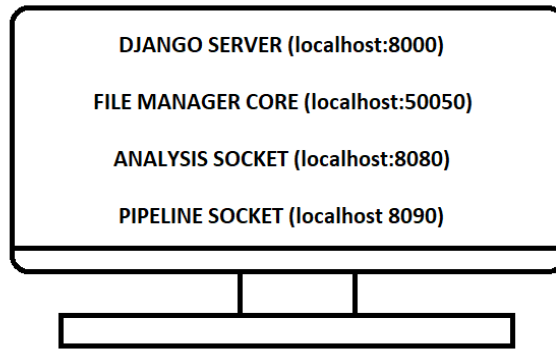


Figure 38: Ports organization working in a localhost environment

Preliminary operations to be executed are:

- 1- Run on **Django Server**, executing the following command from prompt in project folder:

```
python manage.py runserver localhost:8000
```

- 2- **File Manager Core** is constantly running as a Windows Service, hence its ports are always available.
- 3- Run **Analysis and Pipeline sockets** scripts to make the machine listen to the specific ports 8080 and 8090 (by default)
- 4- Run **Celery worker** and **Celery beat** to allow tasks queue management and scheduled processes execution, using the following commands from prompt in project folder:

```
celery -A BQCloudView worker
```

```
celery -A BQCloudView beat
```

After executing preliminary operations, navigate to localhost:8000 from web browser and the following interface will be displayed, prompting for user login.

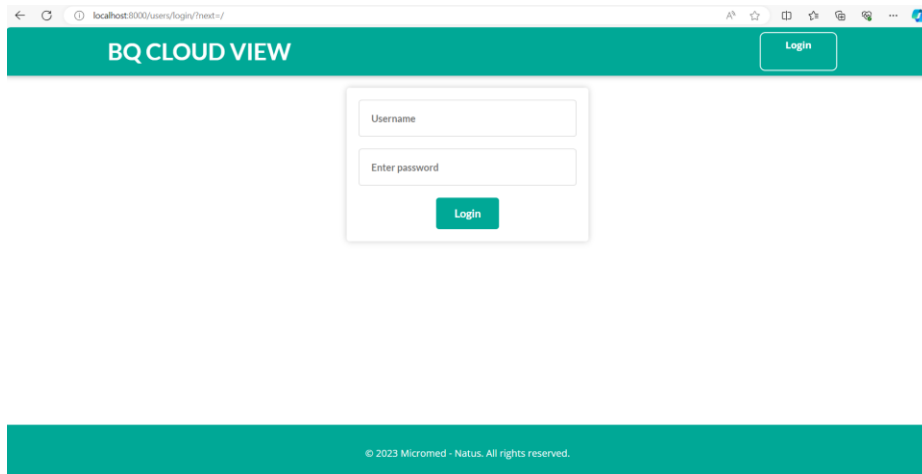


Figure 39: BQ Cloud View Login page

After logging with a valid user, the home page will appear, displaying the three latest exams exported. The exams are kept in synchronization with File Manager Core, which passes all needed information for patients, exams and files.

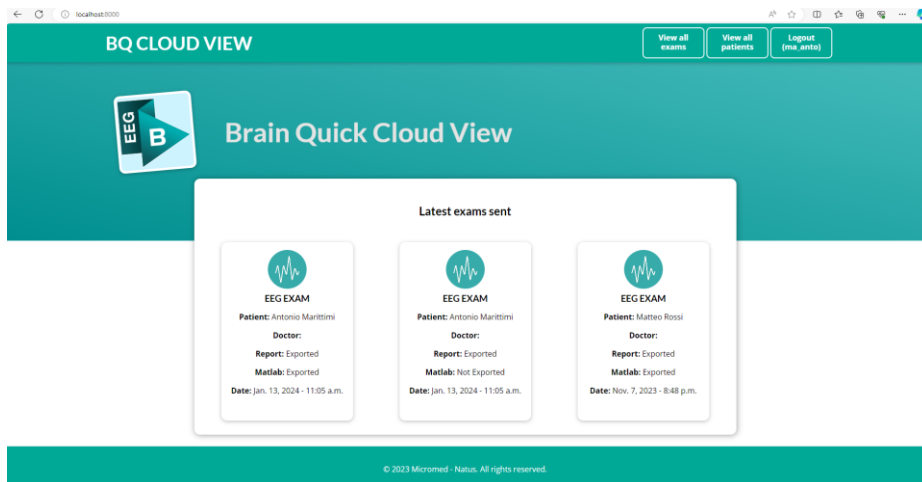


Figure 40: BQ Cloud View home page

The user can then navigate to a single exam or can visualize all available patients and exams using “View all exams” and “View all patients” buttons. From patients and exams’ view, the user can order the objects by export date or patient name.

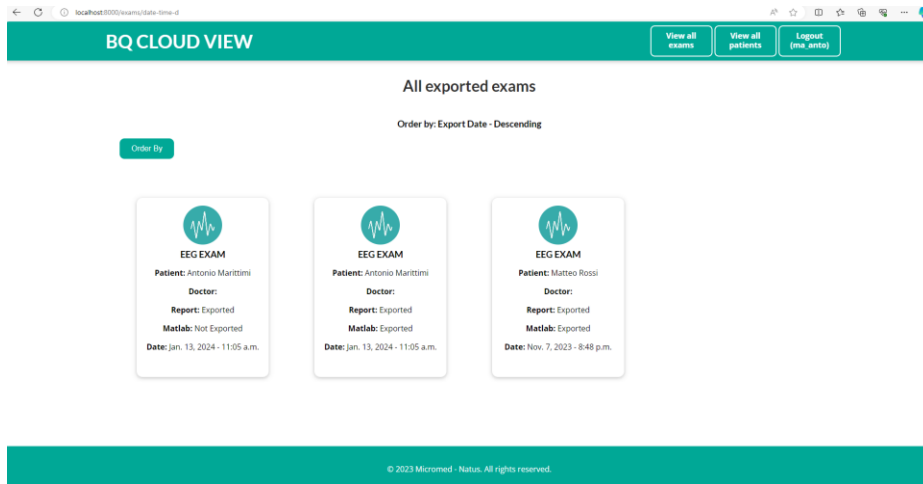


Figure 41: BQ Cloud View available exams' view

Clicking on a specific exam, its information is loaded, together with the possibility of downloading Report, Matlab file, see and start an analysis for the that exam.

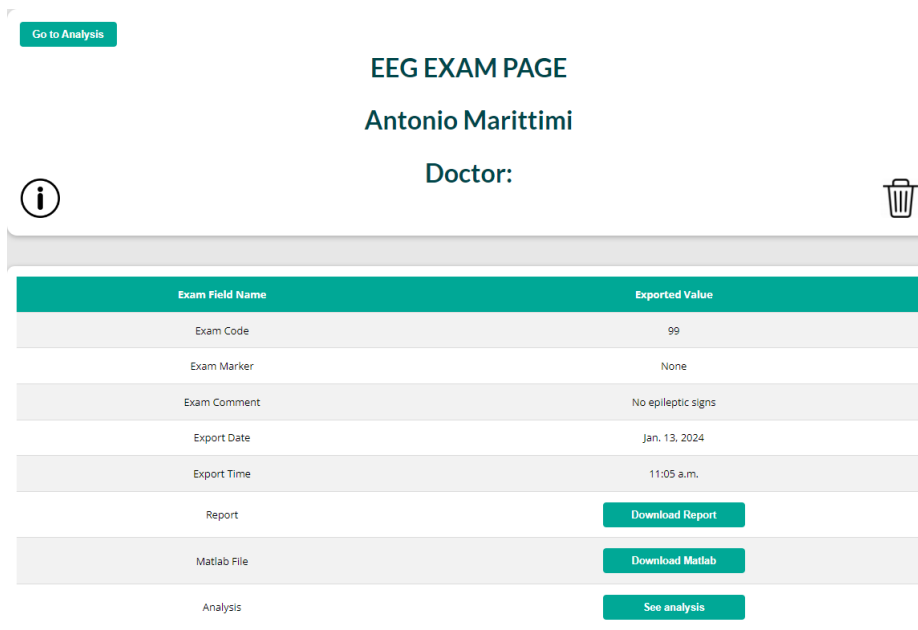


Figure 42: BQ Cloud View exam details

Clicking on “Delete” icon, the user can delete from database the selected exam, while clicking on “Information” button related patient’s page is loaded.

PATIENT PAGE

Antonio Marittimi

Birthdate: 15/02/2001




Available Exams:

Exam ID	Exam Comment	Export Date	Download Report	Download Matlab	Delete Exam
3	No epileptic signs	Jan. 13, 2024	Download	Download	
30	No particular cognitive difficulties highlighted	Jan. 13, 2024	Download	Not Exported	

Figure 43: BQ Cloud View Patient's page

Within the patient's page, user can access detailed patient information by selecting the "Patient" icon. Additionally, he has the option to remove both the patient and associated exams from the database using the "Delete" icon. Furthermore, user can explore all accessible exams related to the selected patient. This includes the ability to download associated report and Matlab files, as well as the option to delete the specific exam.

PATIENT INFORMATION: Antonio Marittimi

Patient Field Name	Exported Value
ID1	11
ID2	MTRRSS01T155123P
Last Name	Marittimi
First Name	Antonio
Birthdate	15/02/2001
Gender	Male
Height	190
Weight	88
Address	None
City	Treviso
Country	Italy
Hospital Department	Neurology
Marker	None
Doctor	Antonio Marittimi
Comment	Not particular pathologies highlighted

Figure 44: BQ Cloud View patient's details

Going back and selecting an exam, user can access analysis module and modify analysis parameters or request a new analysis or pipeline execution by clicking on “Go to Analysis” button.

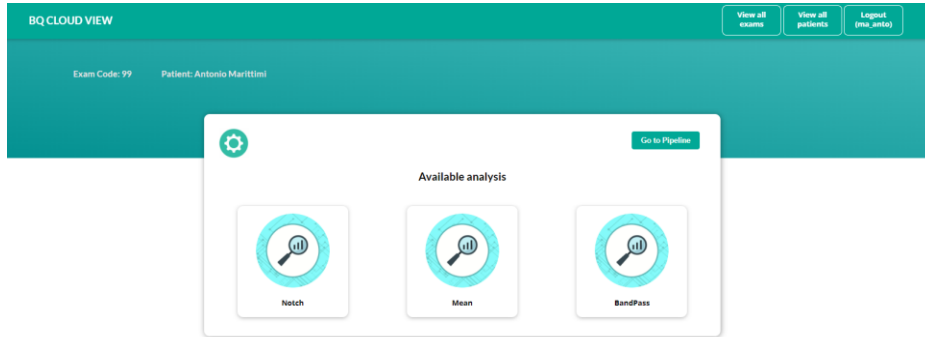


Figure 45: BQ Cloud View analysis page

Clicking on settings icon, user can modify analysis parameters, such as cutoff frequencies for Bandpass filtering.

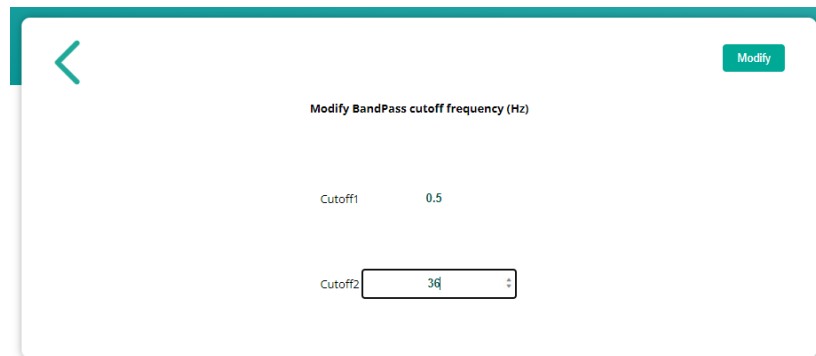


Figure 46: BQ Cloud View Analysis settings

Returning to the analysis page, user can initiate an analysis by clicking on the desired one. Initiating an analysis involves the request being handled through the Celery queue and subsequently redirected to localhost:8080, responsible for analysis execution. The analysis is then processed via the analysis socket, and the request returns to localhost:8000, adding the analysis to the respective exam. This information becomes visible from the exam details page.

Request sent, analysis will be soon available

EEG ANALYSIS PAGE

Patient: Antonio Marittimi

Exam Code: 99

Go to Exam

Analysis ID	Type	Date	Time	Report Analysis	Matlab Analysis	Delete
99-Notch-0	Notch	Nov. 25, 2023	12:15 p.m.	Download	Download	✕
99-Pipeline	Pipeline	Nov. 25, 2023	12:15 p.m.	Download	Download	✕
99-BandPass	BandPass	Nov. 25, 2023	12:17 p.m.	Download	Download	✕

Figure 47: BQ Cloud View analysis details page

From analysis details page, user can download report generated from analysis, download the Matlab file that has been processed or delete an existing and already executed analysis.

Going back to analysis page once again, user is able of executing or saving a pipeline, a set of up to three analyses to be processed in a row, by clicking on “Go to Pipeline”.

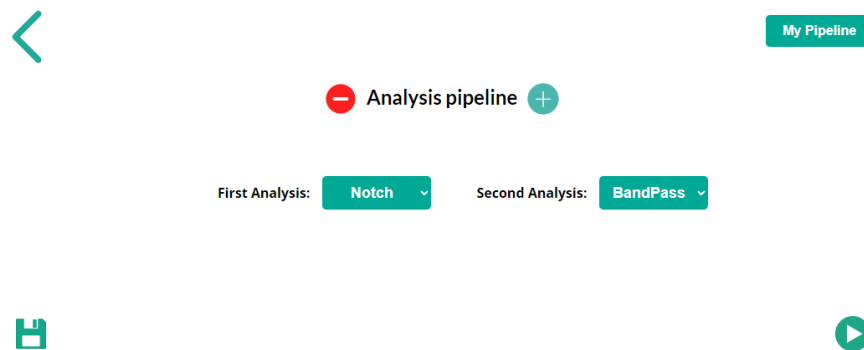


Figure 48: BQ Cloud View analysis pipeline page

After selecting the analysis to be processed, user can save the pipeline from “Save” icon. That specific pipeline will be available on “My Pipeline” to be executed easily in future. Alternatively, user can directly start a pipeline execution from “Play” icon. Initiating a pipeline involves the request being handled through the Celery queue and subsequently redirected to localhost:8090, responsible for pipeline execution. The pipeline is then processed via the pipeline socket, and the request returns to localhost:8000, adding the pipeline (treated exactly as an analysis) to the respective exam. This information becomes visible from the exam details page, as discussed previously.

5. Results: platform deployment in CloudVeneto

This chapter will provide a comprehensive examination of the installation process (Brain Quick Software, SQL EXPRESS and Django Server) on the CloudVeneto platform, along with additional details about the environment. The aim is to establish a functional working environment to be tested with dummy data.

5.1 Client setup

5.1.1 SQL Express installation

On the computer where the data is physically stored (client machine), it's important to set up an SQL engine. This ensures proper storage of data using Micromed standard databases. The recommended engine is SQL Express 2019 or a newer version, which can be obtained by downloading and installing it from the Microsoft website.

On client machine, simply run **SQLEXP_x64_ENU.exe** installer and follow the procedure for installation. After the setup is completed, a local instance of SQL Server will be correctly working, allowing to create local Micromed databases where to store data in.

5.1.2 Brain Quick Software Installation

Brain Quick Software can be installed running the following install shields as administrator on client machine:

- File Manager
- File Manager Core
- Micromed Suite
- Brain Quick EEG
- Brain Quick Acquisition

Installation procedure can be run with default paths and options. After every installation task has been completed and the software license is correctly loaded, client machine is capable of navigating into File Manager archive, import files, open EEG and Video EEG traces and acquire new EEG files.

Since the installation has been successful, File Manager Core service is now listening on default address **IPAddress:50050** for upcoming API requests about resources and database.

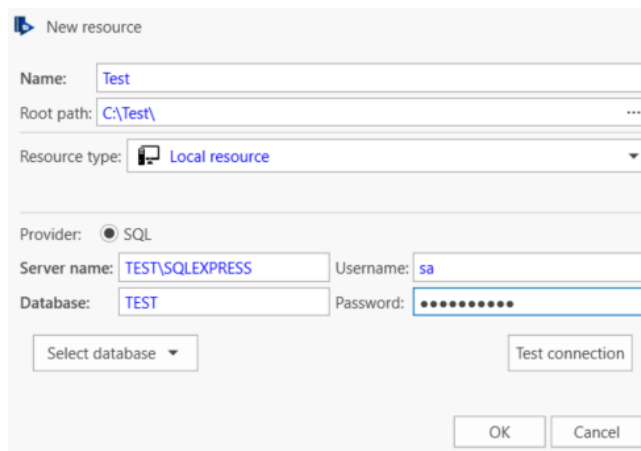
Database and Resource creation

To set up a new repository for importing, copying, or acquiring data, it's fundamental to create a new database and a corresponding folder within the File System for data storage purposes. This process involves utilizing the Micromed tool **SQLSys98Manager**, which necessitates installation on the local machine. Subsequently, launching the application and selecting "New SystemPLUS Database" initiates the setup process.

After compiling the existing fields with Server Name (chosen during SQL EXPRESS installation), Database Name, Authentication User and Root directory, clicking on OK creates a new database in the existing SQL EXPRESS instance.

The last step is to create a new File System folder where to store data linked to newly created database. Folder can be simply created in "C:\\" path.

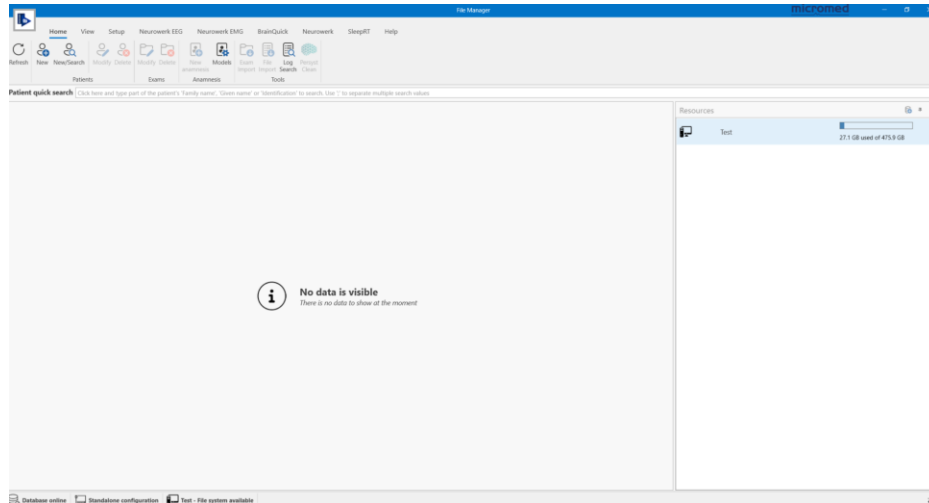
Eventually, launching File Manager allows to create a new resource. After startup, a warning will appear reporting that no resources are available: opting to create a new one opens set up window which needs to be filled with database and folder information just defined:



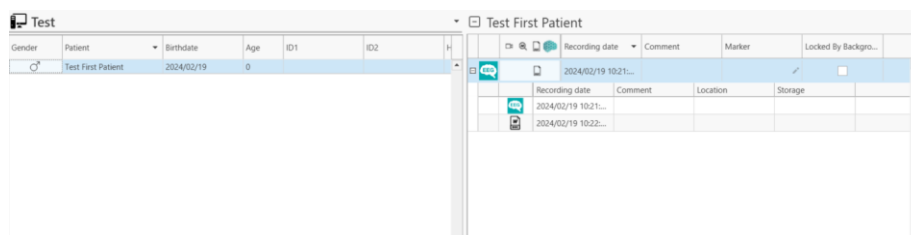
The screenshot shows a "New resource" dialog box with the following fields and options:

- Name: Test
- Root path: C:\Test\
- Resource type: Local resource
- Provider: SQL (selected)
- Server name: TEST\SQLEXPRESS
- Username: sa
- Database: TEST
- Password: [masked]
- Buttons: Select database, Test connection, OK, Cancel

Once the resource has been defined, File Manager interface opens allowing full set of functions:



To test synchronization with Django server automatic task, click on **New Patient** button on Home Tab to create a new patient, then go on Brain Quick tab and choose **Select Protocol**. Run the acquisition with default protocol (that is the simulator), save some minutes of EEG simulated signal and stop the recording. After that, create a new report using **New Report** button from Brain Quick tab on the newly created exam and save it. Once exam with related report has been created, interface will look like the following image:



The expectation is that, once Django server and related tasks have been started, local database will be synchronized with server database, adding patient, exam, and report information to be navigated from web browser.

5.2 Server setup

Setting up the Brain Quick Cloud server involves transferring files to a server machine that already has the Python and Django environment installed, running on a Linux operating system. All the files and folder structures are replicated into a designated folder on the server machine.

To ensure the proper and seamless functioning of Brain Quick Cloud, certain components must be installed on the server machine:

- **Python** (version 3.11 or higher)
- **Django** package
- **Django Rest Framework** package
- **RabbitMQ** server
- **Celery** package
- **Celery Beat** package

5.2.1 Configuration of communication with worker machine

Navigating into project folder and opening **utils.py** file allows to set up parameters for worker parameters' configuration, IP Address and Port specifically. In this setup, worker machine's parameters need to be set to communicate with the virtual worker machine. The default ports for Analysis and Pipeline execution are set to 8080 and 8090 respectively, and they're hardcoded into the project's code and on the sockets created in worker machine.

```
url_remoto = "http://localhost:8080/" #worker IP and PORT
configuration for analysis
url_remoto = "http://localhost:8090/" #worker IP and PORT
configuration for workflow
```

5.2.2 Automatic Task settings for data detection

Navigating into project folder and opening **tasks.py** file allows to set up parameters for automatic task for data detection on Client machine database.

```

@shared_task
def exams_discovery_task():

    try:
        request_host = "localhost"
        host = "localhost"
        port = "8000"
        response_resources = requests.get("http://" +
request_host + ":50050/api/Patient/resources") #look for all
available resources

```

host and **port** must be the ones used for Django server running and allow for data storage into server, while **request_host** must be the one of client machine, where File Manager Core (and hence Brain Quick Software) is installed, as you can see from the subsequent GET method to retrieve resources from Web API.

5.2.3 Server Run and Port Forwarding

Once you've successfully installed all the packages and files, the next step is to run the server. Specify its unique IP address and port and run the command from command prompt locating in the **settings.py** file folder to get the server up and running, as discussed, some paragraphs above (host and port must be the same set above).

python manage.py runserver IPADDRESS:PORT

Thanks to port forwarding tool provided by CloudVeneto, a machine connected via SSH to cloud's gateway can easily access the server by typing in web browser **IPADDRESS:PORT** and hence can gain full access to Brain Quick Cloud interface. After server is online, it is necessary to manually start Celery task queue and Celery beat automatic task executer with the commands specified in paragraph 4.4.

5.2.4 Database initialization

Once the server runs up, the database **db.sqlite3** is created and initialized with default tables contained in **models.py** settings file. The next step is to create:

- a **superuser** that will have complete access to Django project's admin area
- **user groups** (doctor and patient)
- a set of **test users** to authenticate on server
- **default analysis types and parameters**, which execution can be requested from server

Ensure the proper configuration of the newly initialized database by referring to the steps outlined in the **Useful Documents** section of this document, specifically in Chapter 6.

5.3 Worker setup

Setting up the Worker machine means copying all python files useful for web socket creation for request exchange with Django server, as well as modules for analysis and pipeline executions. Python package (3.11 or later) needs to be installed to run the code scripts allowing for web communication.

Files to be copied are:

- **socket_analysis.py**
- **socket_pipeline.py**
- **analysis_module.py**
- **pipeline_module.py**

The first two files allow for socket creation respectively in ports 8080 and 8090 and need to be run at Virtual machine startup, while the second set of files contain all the logic necessary to convert data from Matlab to python, to process the requested analysis, reconvert data to Matlab standards and generate a summarized report on the analysis executed. Moreover, from **socket_analysis.py** and **socket_pipeline.py** it is important to set up the absolute paths where data are locally saved once the request with EEG data has arrived from Django server and once the analysis has been executed. Moreover, it is necessary to change **HOST_server** and **PORT_server** parameters, using IP Address and port of Django server that is running up.

For `socket_analysis.py`:

```
new_folder_filtering =
'C:\\Users\\Public\\Documents\\Micromed\\BQCloudView\\Analysis\\Mean'
new_folder_mean =
'C:\\Users\\Public\\Documents\\Micromed\\BQCloudView\\Analysis\\Notch'
new_folder_spectrum =
'C:\\Users\\Public\\Documents\\Micromed\\BQCloudView\\Analysis\\Band Pass'
new_folder_executed =
'C:\\Users\\Public\\Documents\\Micromed\\BQCloudView\\Analysis\\Executed'

HOST_server = "localhost"
PORT_server = "8000"
```

For `socket_pipeline.py`:

```
new_folder_pipeline =
'C:\\Users\\Public\\Documents\\Micromed\\BQCloudView\\Pipeline\\'
new_folder_pipeline_executed =
'C:\\Users\\Public\\Documents\\Micromed\\BQCloudView\\Pipeline\\Executed'

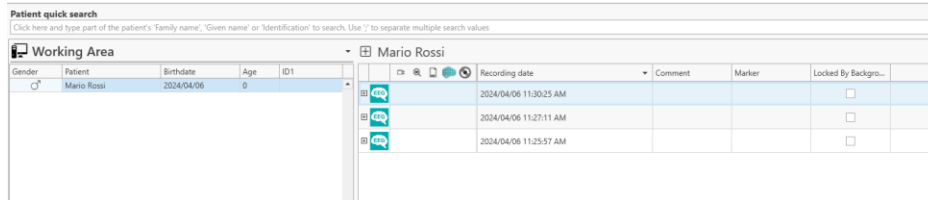
HOST_server = "localhost"
PORT_server = "8000"
```

5.4 Validation and testing

Once all machines and configuration files have been properly set with addresses, ports and paths, before testing the entire environment it is mandatory to follow the

steps contained in **Useful Documents** section, to make sure the database is correctly set up and ready to be tested.

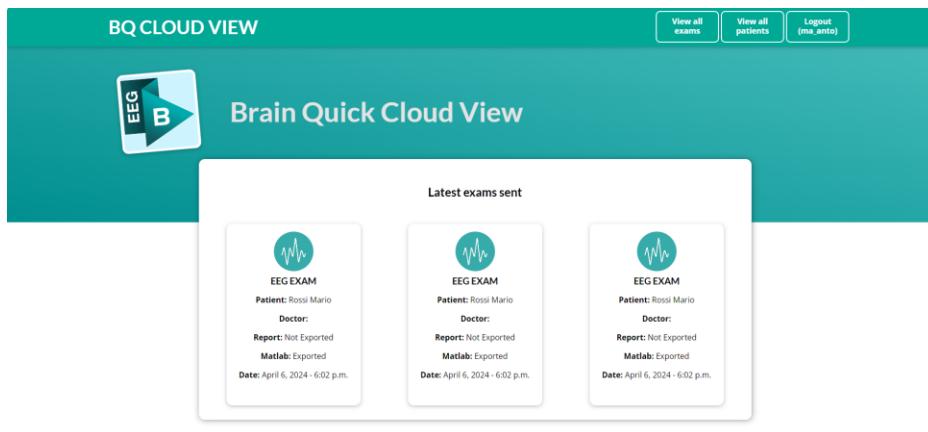
A new patient has been registered on the client machine, and several EEG examinations have been conducted. These examinations are now prepared for synchronization with a Django server. The task responsible for discovering these examinations runs periodically on the server machine, allowing the latest patient and exam information is reflected in the server's database.



Patient quick search
Click here and type part of the patient's 'Family name', 'Given name' or 'Identification' to search. Use ";" to separate multiple search values

Working Area Mario Rossi

Gender	Patient	Birthdate	Age	ID1	Recording date	Comment	Marker	Locked By Background
♂	Mario Rossi	2024/04/06	0		2024/04/06 11:30:25 AM			<input type="checkbox"/>
					2024/04/06 11:27:11 AM			<input type="checkbox"/>
					2024/04/06 11:25:57 AM			<input type="checkbox"/>



BQ CLOUD VIEW [View all exams](#) [View all patients](#) [Logout \(ma_sirto\)](#)

Brain Quick Cloud View

Latest exams sent

- EEG EXAM**
Patient: Rossi Mario
Doctor:
Report: Not Exported
Matlab: Exported
Date: April 6, 2024 - 6:02 p.m.
- EEG EXAM**
Patient: Rossi Mario
Doctor:
Report: Not Exported
Matlab: Exported
Date: April 6, 2024 - 6:02 p.m.
- EEG EXAM**
Patient: Rossi Mario
Doctor:
Report: Not Exported
Matlab: Exported
Date: April 6, 2024 - 6:02 p.m.


It will be possible to download report and Matlab file, or to request an analysis/pipeline on the selected exam.

Exam Field Name	Exported Value
Exam Code	
Exam Marker	
Exam Comment	
Export Date	April 6, 2024
Export Time	6:02 p.m.
Report	Not Exported
Matlab File	Download Matlab
Analysis	See analysis


Exam ID: 9 Patient: Rossi Mario

[Go to Pipeline](#)


Available analysis



Mean



Notch




BandPass

Once the analysis has been completed, data are sent back to server machine, allowing to download the cleaned data and the report containing analysis parameters.

EEG ANALYSIS PAGE

Patient: Rossi Mario
Exam Code: 9

[Go to Exam](#)

Analysis ID	Type	Date	Time	Report Analysis	Matlab Analysis	Delete
mario-rossi-bandpass	BandPass	April 6, 2024	6:21 p.m.	Download	Download	

6. Documentation

This chapter will provide basics operations to be performed after a new database has been created, to make sure it is properly initialized and ready to be used. The next subchapters include superuser creation, user groups and users' database creation, as well as analysis types creation.

Once the server has been run for the very first time, in project folder an empty **db.sqlite3** database is created. It will be crucial to run the migrations already implemented in project's code to allow proper database structure creation. The operation can be performed by stopping the running server and by typing the following command:

```
python manage.py migrate
```

Database is now populated with the correct tables.

6.1 Superuser

After database has been initialized, it is fundamental to define a superuser, that will gain access to embedded Django admin panel in administrator mode. The operation can be executed by running the following command, always on command prompt after changing directory to project's folder:

```
python manage.py createsuperuser
```

Following the subsequent steps on prompt, a new superuser will be created. It is now possible to start the server and to login into Django administration panel, useful to configure groups, users and analysis types. To access Django admin panel it is sufficient to type the following URL in web browser:

```
IPADDRESS:PORT/admin
```

Where IPADDRESS and PORT are the ones defined in server run command. The user for authentication is the one just registered from command prompt.

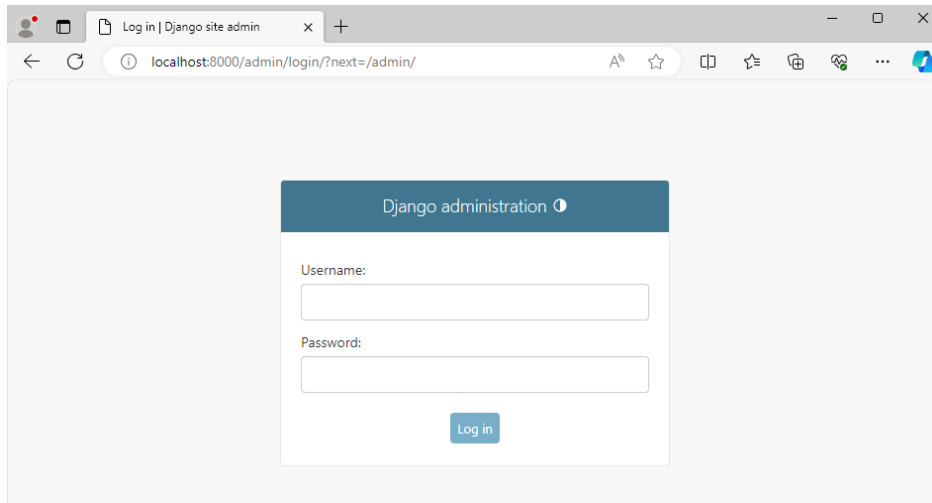


Figure 49: Django administration panel authentication

After logging in, the administrator accesses to the Dashboard and has the possibility to access and modify **Authentication and Authorization** section to add groups and users, and to modify **PatientCloudView** section, with all operations needed to initialize project's database.

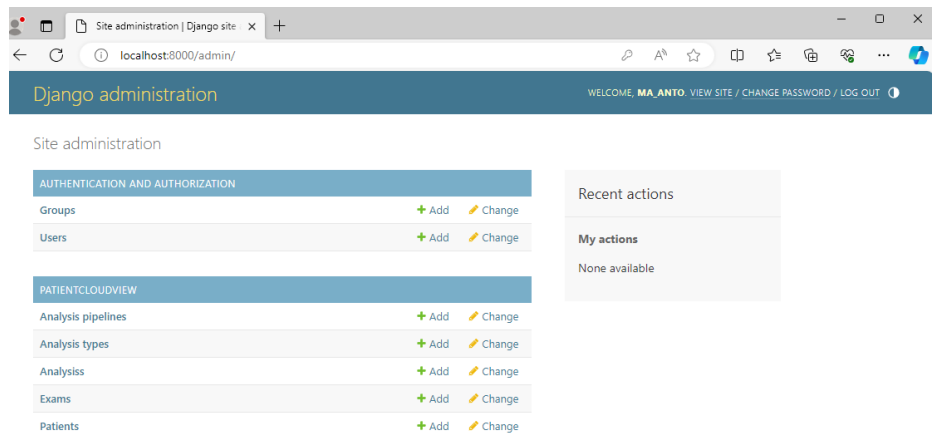
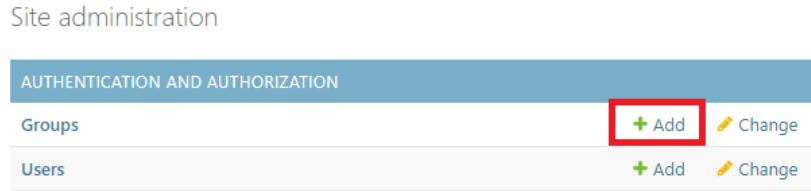


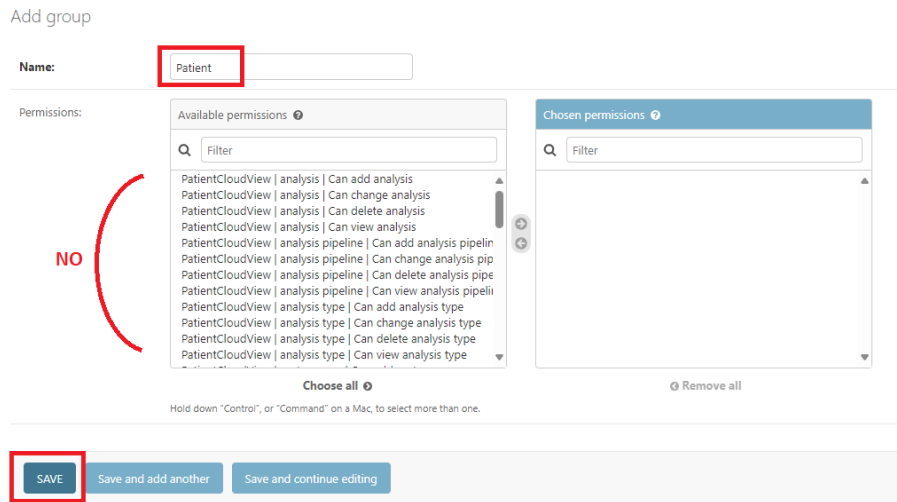
Figure 50: Django administration panel dashboard

6.2 User groups

From administrator dashboard, click on **Add** button in **Groups** section:



It is then necessary to create **Patient** and **Doctor** groups, that are managed from BQ Cloud View project to differentiate read, write and visualization permissions. To create a group, simply type group name and save it, without assigning default permissions provided by Django, since they are directly managed from different project views.



After adding both groups they shall be visible in **Groups** section:

AUTHENTICATION AND AUTHORIZATION	
Groups	+ Add
Users	+ Add
PATIENTCLOUDVIEW	
Analysis pipelines	+ Add
Analysis types	+ Add
Analysis	+ Add
Exams	+ Add
Patients	+ Add

Select group to change

Q Search

Action: ----- Go 0 of 2 selected

GROUP

Doctor

Patient

2 groups

6.3 Users

Once the groups have been defined, clicking on **Add** button in Users section allows creation of users based on **Username**, **Password** and **Slug ID**; the latter will be useful to match users (patients specifically) with their related exams. While Username and Password may be defined without constraints, Slug ID needs to be defined in the format **FirstName-LastName-Birthdate** (DDMMYYYY) to match information coming from File Manager Core APIs. Slug ID needs to be defined even for the first created Superuser:

AUTHENTICATION AND AUTHORIZATION	
Groups	+ Add
Users	+ Add
PATIENTCLOUDVIEW	
Analysis pipelines	+ Add
Analysis types	+ Add
Analysis	+ Add
Exams	+ Add
Patients	+ Add

Add user

First, enter a username and password. Then, you'll be able to edit more user options.

Username:
Required: 1-50 characters or fewer. Letters, digits and @/./+/-/_ only.

Password:
Your password can't be too similar to your other personal information.
Your password must contain at least 8 characters.
Your password can't be a commonly used password.
Your password can't be entirely numeric.

Password confirmation:
Enter the same password as before, for verification.

CUSTOM USER

Custom user: #1

Slug id:

SAVE Save and add another Save and continue editing

When clicking on **Save**, user is added to users table and it is possible to assign the belonging group, depending on user's category (doctor or patient) for server authentication.

6.4 Analysis

Default analyses discussed in previous chapters are initialized together with the database. The main feature of this project is that users can add and manage additional types of analysis to be performed on EEG data. This subchapter aims to explain how to insert into project's code a new analysis to be managed singularly or with a pipeline.

6.4.1 Adding the analysis type to database

The very first step consists of adding the analysis type to server's database, from Django administration dashboard. After logging with the superuser account, simply click on Add button on **Analysis Types** section and fill the information about **Analysis Name** and **Analysis Parameters**.

Site administration

The screenshot shows the Django administration interface. It features two main sections: 'AUTHENTICATION AND AUTHORIZATION' and 'PATIENTCLOUDVIEW'. The 'PATIENTCLOUDVIEW' section contains a table with the following items:

AUTHENTICATION AND AUTHORIZATION	
Groups	+ Add ✎ Change
Users	+ Add ✎ Change

PATIENTCLOUDVIEW	
Analysis pipelines	+ Add ✎ Change
Analysis types	+ Add ✎ Change
Analysisiss	+ Add ✎ Change
Exams	+ Add ✎ Change
Patients	+ Add ✎ Change

In the 'PATIENTCLOUDVIEW' table, the '+ Add' button for 'Analysis types' is highlighted with a red rectangular box.

While the name is a text field, parameters are saved in JSON format. It is fundamental to fill parameters as shown below ("**Parameter_Value**" :

“**Parameter_Name**”). Once all parameters have been defined for the customized analysis, click on Save to add the analysis to database.

Change analysis type

BandPass

Analysis Name:

Analysis Parameters:

6.4.2 Adding the analysis type to pipeline form

Navigate into **forms.py** project file and search for **PipelineForm** form object. It will be sufficient to add the newly created analysis to multiple choice form for each of the three analyses.

```
First_Analysis = forms.ChoiceField(choices=[('None', 'None'), ('Mean', 'Mean'), ('Notch', 'Notch'), ('BandPass', 'BandPass'), ('New_Analysis_Name', 'New_Analysis_Name')], label="First Analysis")

Second_Analysis = forms.ChoiceField(choices=[('None', 'None'), ('Mean', 'Mean'), ('Notch', 'Notch'), ('BandPass', 'BandPass'), ('New_Analysis_Name', 'New_Analysis_Name')], label="Second Analysis")

Third_Analysis = forms.ChoiceField(choices=[('None', 'None'), ('Mean', 'Mean'), ('Notch', 'Notch'), ('BandPass',
```

```
'BandPass'), ('New_Analysis_Name', 'New_Analysis_Name')],  
label="Third Analysis")
```

Analysis pipeline

First Analysis:

None ▾

- None
- Mean
- Notch
- BandPass
- New_Analysis_Name

6.4.3 Adding the analysis type to analysis and pipeline modules on worker machine

As we already discussed, modules on worker machine are responsible for data processing based on the analysis type chosen from the user. Hence, it will be crucial to add the analysis algorithm to both analysis and pipeline modules.

Analyses are simply python functions that require variables such as Matlab file, analysis parameters, ecc. (see Bandpass function from analysis module below).

```
def BandPass(matlab, PatientSlug, parameters, path_save):
```

Each function exploits a **mat2python** procedure that allows conversion from Micromed Matlab structure format to list of dictionaries, allowing an easier data management. See the following code snippet to understand how to build the starting point for your own analysis, getting EEG data matrix and parameters from request to worker machine:

```
def BandPass(matlab, PatientSlug, parameters, path_save):  
  
    cutoff1 = float(parameters [0])  
    cutoff2 = float(parameters [1])
```

```

EEG_data = mat2python(matlab)
Fs = EEG_data['sampling_frequency']
EEG_data['Analysis'] = "Bandpass with cutoff1 = " +
str(cutoff1) + " Hz" + " and cutoff2 = " + str(cutoff2)
EEG_matrix = EEG_data['EEG_matrix']
number_of_channels = EEG_data['number_of_channels']

```

EEG_matrix will be the starting point for data processing, containing the matrix with EEG samples over time samples (**number of samples x number of channels**). The customized analysis may be built taking the other analysis as code reference.

After the signal has been elaborated with the customized algorithm, **EEG_matrix** will be replaced from the processed EEG data and the python data format converted back to a Matlab structure to be sent back to Django server.

```

EEG_data['EEG_matrix'] = bandpass_matrix

```

```

s.savemat(path_save + '\\Executed\\' + str(PatientSlug) +
'.mat', {'EEG': EEG_data})

```

The new analysis function needs to be added both to **analysis_module.py** and **pipeline_module.py**.

6.4.4 Adding the analysis type to analysis and pipeline sockets on worker machine

The very last step is adding the logic necessary to call the desired analysis function on **socket_analysis.py** and **socket_pipeline.py**, which are the scripts responsible for collecting analyses and pipeline requests and for processing them.

```

ef Analysis(matlab, ExamSlug, PatientSlug, Analysis_Name,
parameters, counter):

    global HOST_server
    global PORT_server

```

```
if Analysis_Name == "Mean":
    analysis_module.Mean(matlab, PatientSlug, path_save,
counter)

if Analysis_Name == "Notch":
    analysis_module.Notch(matlab, PatientSlug,
parameters, path_save, counter)

if Analysis_Name == "BandPass":
    analysis_module.BandPass(matlab, PatientSlug,
parameters, path_save, counter)

if Analysis_Name == "New_Analysis_Name":
    analysis_module.New_Analysis_Name(matlab,
PatientSlug, parameters, path_save, counter)
```

In the code snippet displayed above, **New Analysis** is added to the list of available ones and thus can be processed in case of incoming request.

7. Conclusion and future directions

This chapter briefly describes what has been done on CloudVeneto platform to switch from a development environment to a real one.

7.1 Virtual machines configuration

First of all, from CloudVeneto admin dashboard, we accessed to project environment, named D4C (Diagnostic For Change), instancing three virtual machines. The first two ones (Server and Worker) are Unix based, while the third one (Client) needs to be Windows based to install Micromed Softwares.

Then, the three machines have been configured separately, as already discussed in the previous chapters (i.e., Django, File Manager and Python configuration). Everything has been done connecting via SSH to the machines or via remote desktop in case of Windows based one, to copy script files and modify them to adapting IP addresses and Ports.

Starting Celery worker and beat scheduler on server and web sockets on worker then, allowed central Django database to be synchronized with File Manager database, and using port forwarding tool provided by CloudVeneto, we were able to access interface rendered by the server from our web browser, accessing data got from File Manager and requesting analysis algorithm execution on EEG data recorded in Brain Quick.

7.3 Deployment example

These final tests conclude the project scope and leave on CloudVeneto platform a working environment, ready to be used and improved, implementing additional features or modifications directly on source code, following the future improvements described in the next subchapters.

Below, some screenshots taken from client, server and worker machine after deployment on CloudVeneto platform:

```
ubuntu@d4c-worker: ~/test_remote_machine
ubuntu@d4c-worker:~/test_remote_machine$ python socket_analysis.py
Server in ascolto sulla porta 8080
file ricevuti saranno salvati in: /home/ubuntu/BQCloudView/Analysis/
```

Figure 51: Worker machine, socket for analysis listening on port 8080

```
ubuntu@d4c-worker: ~/test_remote_machine
ubuntu@d4c-worker:~/test_remote_machine$ python socket_pipeline.py
Server in ascolto sulla porta 8090
file ricevuti saranno salvati in: /home/ubuntu/BQCloudView/Pipeline/
```

Figure 52: Worker machine, socket for pipeline listening on port 8090

```
ubuntu@d4c-django: ~/D4C
ubuntu@d4c-django:~/D4C$ celery -A BQCloudView beat
celery beat v5.3.6 (emerald-rush) is starting.
...
LocalTime -> 2024-04-17 12:57:35
Configuration ->
. broker -> amqp://guest:**@localhost:5672//
. loader -> celery.loaders.app.AppLoader
. scheduler -> celery.beat.PersistentScheduler
. db -> celerybeat-schedule
. logfile -> [stderr]@WARNING
. maxinterval -> 5.00 minutes (300s)
```

Figure 53: Server machine, Celery beat started

```

ubuntu@d4c-django: ~/D4C
*****
Linux-5.15.0-100-generic-x86_64-with-glibc2.35 2024-04-17 13:05:23
***
** [config]
** .> app: BQCloudView:0x7f4b16cf8340
** .> transport: amqp://guest:**@localhost:5672//
** .> results: disabled://
** .> concurrency: 2 (solo)
** .> task events: OFF (enable -E to monitor tasks in this worker)
*****
[queues]
.> celery exchange=celery(direct) key=celery

[2024-04-17 13:05:23,746: WARNING/MainProcess] No hostname was supplied. Reverting to default 'localhost'
[2024-04-17 13:05:24,817: WARNING/MainProcess] Richiesta PUT riuscita! Paziente già esistente e aggiornato
[2024-04-17 13:05:24,903: WARNING/MainProcess] Richiesta PUT riuscita! Esame già esistente e aggiornato
[2024-04-17 13:05:24,915: WARNING/MainProcess] Richiesta PUT riuscita! Paziente già esistente e aggiornato
[2024-04-17 13:05:24,997: WARNING/MainProcess] Richiesta PUT riuscita! Esame già esistente e aggiornato
[2024-04-17 13:05:25,008: WARNING/MainProcess] Richiesta PUT riuscita! Paziente già esistente e aggiornato
[2024-04-17 13:05:25,090: WARNING/MainProcess] Richiesta PUT riuscita! Esame già esistente e aggiornato
[2024-04-17 13:05:25,100: WARNING/MainProcess] Richiesta PUT riuscita! Paziente già esistente e aggiornato
[2024-04-17 13:05:25,181: WARNING/MainProcess] Richiesta PUT riuscita! Esame già esistente e aggiornato
[2024-04-17 13:05:25,192: WARNING/MainProcess] Richiesta PUT riuscita! Paziente già esistente e aggiornato
[2024-04-17 13:05:25,272: WARNING/MainProcess] Richiesta PUT riuscita! Esame già esistente e aggiornato
[2024-04-17 13:05:25,284: WARNING/MainProcess] Richiesta PUT riuscita! Paziente già esistente e aggiornato
[2024-04-17 13:05:25,361: WARNING/MainProcess] Richiesta PUT riuscita! Esame già esistente e aggiornato
[2024-04-17 13:05:25,371: WARNING/MainProcess] Richiesta PUT riuscita! Paziente già esistente e aggiornato
[2024-04-17 13:05:25,454: WARNING/MainProcess] Richiesta PUT riuscita! Esame già esistente e aggiornato

```

Figure 54: Server machine, Celery worker started with related logs

```

ubuntu@d4c-django: ~
ubuntu@d4c-django:~$ cd D4C
ubuntu@d4c-django:~/D4C$ python manage.py runserver 10.67.35.136:7000
Watching for file changes with StatReloader
Performing system checks...

System check identified no issues (0 silenced).
April 17, 2024 - 13:05:08
Django version 3.2.12, using settings 'BQCloudView.settings'
Starting development server at http://10.67.35.136:7000/
Quit the server with CONTROL-C.
[17/Apr/2024 13:05:24] "PUT /api/Surname-Name-4122024/patient-update HTTP/1.1" 200 264
[17/Apr/2024 13:05:24] "PUT /api/Surname-Name-4122024/Name-Surname-4122024-1/exam-update HTTP/1.1" 200 302
[17/Apr/2024 13:05:24] "PUT /api/Surname-Name-4122024/patient-update HTTP/1.1" 200 264
[17/Apr/2024 13:05:24] "PUT /api/Surname-Name-4122024/Name-Surname-4122024-1/exam-update HTTP/1.1" 200 302
[17/Apr/2024 13:05:25] "PUT /api/Surname-Name-4122024/patient-update HTTP/1.1" 200 264
[17/Apr/2024 13:05:25] "PUT /api/Surname-Name-4122024/Name-Surname-4122024-1/exam-update HTTP/1.1" 200 302
[17/Apr/2024 13:05:25] "PUT /api/Surname-Name-4122024/patient-update HTTP/1.1" 200 264
[17/Apr/2024 13:05:25] "PUT /api/Surname-Name-4122024/Name-Surname-4122024-1/exam-update HTTP/1.1" 200 302
[17/Apr/2024 13:05:25] "PUT /api/Surname-Name-4122024/patient-update HTTP/1.1" 200 264
[17/Apr/2024 13:05:25] "PUT /api/Surname-Name-4122024/Name-Surname-4122024-1/exam-update HTTP/1.1" 200 302
[17/Apr/2024 13:05:25] "PUT /api/Surname-Name-4122024/patient-update HTTP/1.1" 200 264
[17/Apr/2024 13:05:25] "PUT /api/Surname-Name-4122024/Name-Surname-4122024-1/exam-update HTTP/1.1" 200 302
[17/Apr/2024 13:05:25] "PUT /api/Surname-Name-4122024/patient-update HTTP/1.1" 200 264
[17/Apr/2024 13:05:25] "PUT /api/Surname-Name-4122024/Name-Surname-4122024-1/exam-update HTTP/1.1" 200 302

```

Figure 55: Server machine, Django web server in run with related logs

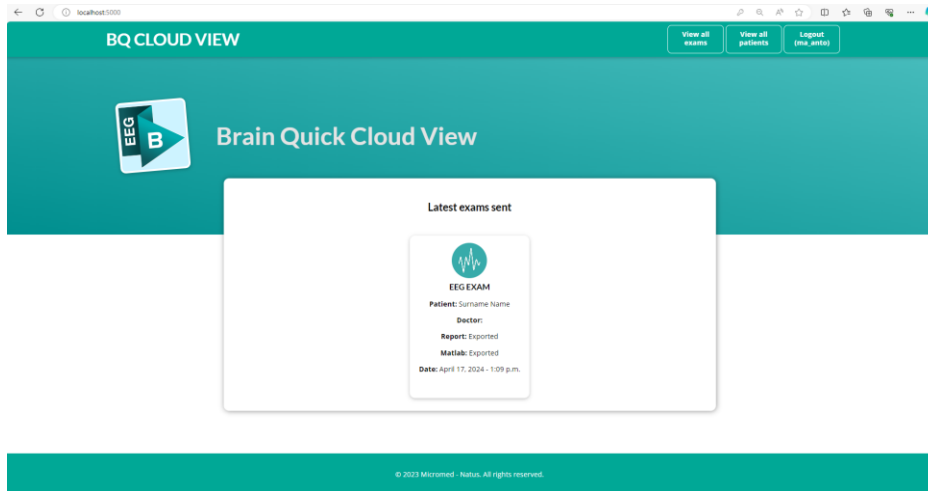


Figure 56: Web browser, latest exams exported

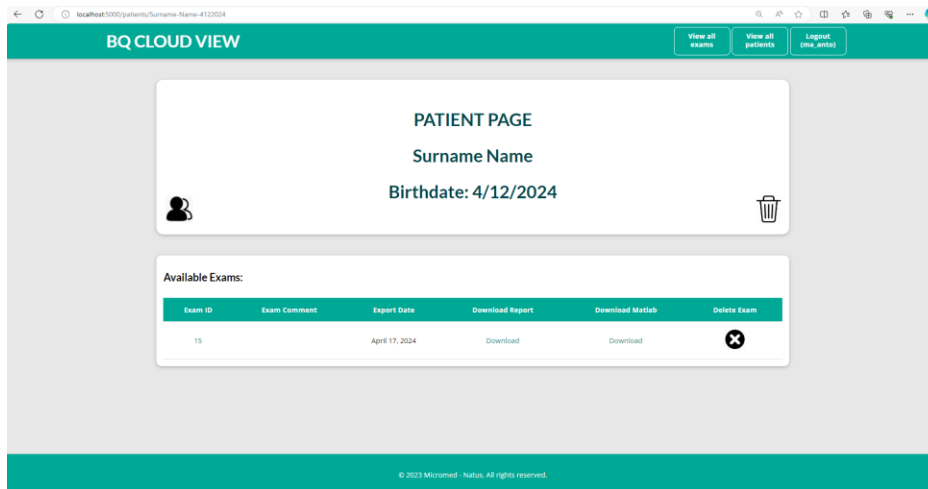


Figure 57: Web browser, exported patient details

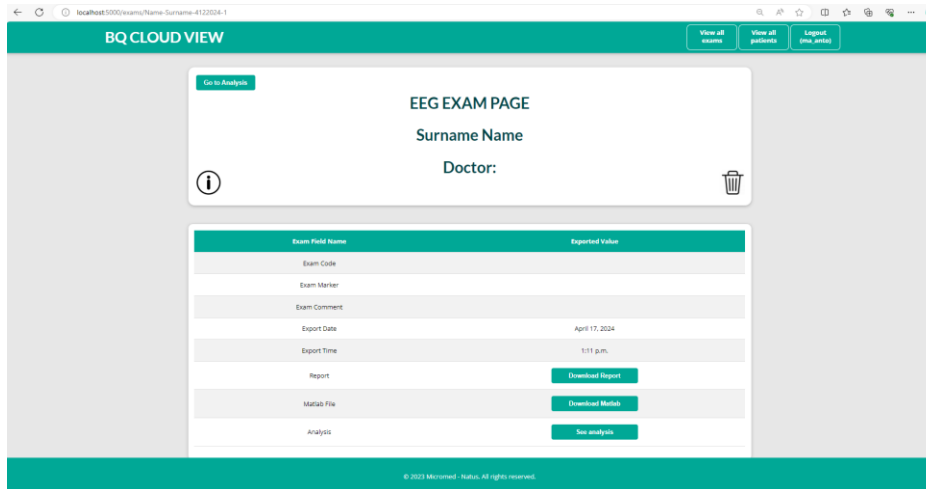


Figure 58: Web browser, exported exam details

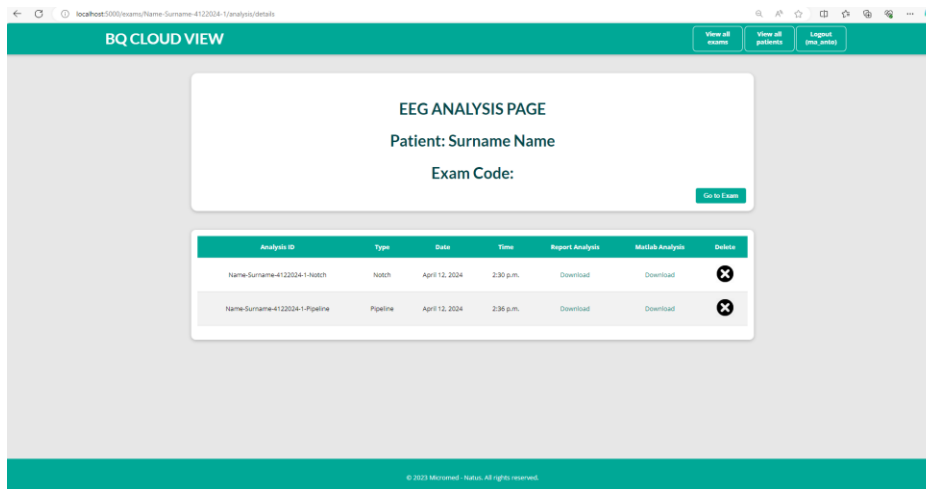


Figure 59: Web browser, analysis executed on the exported exam

7.4 Future improvements

This subchapter aims to outline some essential considerations for future implementations and testing of the system with actual patient data under more complex conditions.

7.4.1 Automate database initialization

After launching the server for the first time, users are responsible for initiating the database setup, which involves creating groups, users, and defining new analysis

types. To facilitate this process, developing an application could be beneficial. Such an application would automate the population of the database with essential tables and connections. Additionally, it could offer a user-friendly interface where users can input their preferences to create groups, users, and analysis types efficiently.

7.4.2 Crypting data transmission

Data transmission and information exchange among Server, Client and Worker machines are totally managed by WebAPIs. This applies to both incoming data from the File Manager Core stored in the server's database and to the traffic going in both directions between the server and the worker machine.

Communication via Web APIs without encryption poses a serious threat to data security. Without adequate protection measures, data transmitted via the API could be intercepted by malicious individuals, putting sensitive information at risk and compromising user privacy. Moreover, data exchanged from a medical device software could be either corrupted or changed, resulting in a wrong diagnosis and a high risk for the patient himself.

To ensure an adequate level of security in Web API communication, it is essential to adopt **encryption**. This technique employs secure protocols such as HTTPS (HTTP over SSL/TLS), which encrypt data during transfer, preventing attackers from accessing it. Furthermore, it is necessary to implement an authentication system to verify that only authorized users can access the data and resources of the API. In addition to encryption and authentication, it is important to follow security practices such as proper credential management, strict validation of input data and implementation of appropriate authorization controls. It is also essential to keep the libraries and dependencies used in the API implementation up to date to mitigate any known vulnerabilities and ensure ongoing protection.

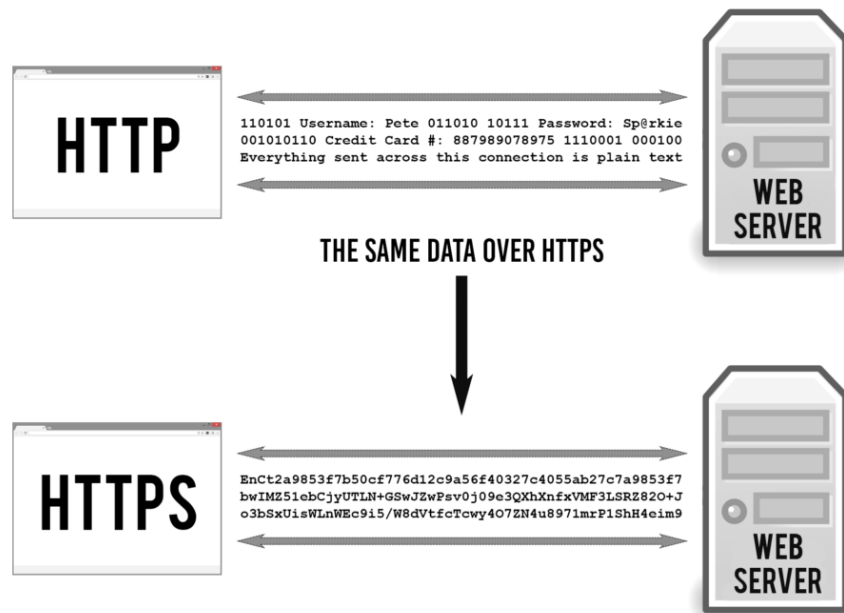


Figure 60: HTTP and HTTPS protocols [22]

The lack of security causes the project to be still private and not accessible from everyone. Working towards this direction to guarantee safe data exchange, may allow this web application to spread and be used in real cloud environments in public way.

7.4.3 Mobile Home Video Telemetry – Test with real patient data

Tests performed so far account for dummy data recorded directly from client machine in CloudVeneto platform to check the correct functioning of the entire system. But in a real and more complex environment, EEG traces come from a patient under recording with Micromed devices and are not saved in cloud platform but in some way sent from the acquisition machine to the local resource persisting in CloudVeneto. How can this situation be managed?

7.4.3.1 Mobile Home Video Telemetry (MHVT) system

MHVT is a product developed by Micromed, consisting of a case with all necessary equipment for recording a real Video EEG recording directly at patient’s home,

allowing him to be more comfortable and so more subject to epileptic attacks or other crisis.

The case essentially contains:

- Acquisition machine with Brain Quick Software (tablet)



Figure 61: Micromed Mobile Home Video Telemetry System [23]

- Micromed acquisition device (MORPHEUS)



Figure 62: MORPHEUS acquisition device for polysomnography [23]

- Video camera to record patient's movements



Figure 63: AXIS M1065-L camera for video recording [23]

- Router to make sure the device is always online
- Power supply cable

The patient undergoes preparation for recording while in the hospital environment and is subsequently discharged home with the necessary equipment. Upon returning home, the patient simply needs to connect the power supply, open the case, and properly position the camera for recording.

7.4.3.2 Data exchange implementation with CloudVeneto

MHVT acquires data in a local resource linked to a SQL database, so how to send data to cloud platform directly from patient's home?

This first requires the implementation of a VPN to make sure that the resource defined in Client machine of CloudVeneto is reachable from the local network of MHVT. Once the tunnel has been defined and configured, it is possible to set up a Microservice embedded in File Manager Core from the acquisition machine, named Background Transfer.

The service simply periodically scans a resource looking for new data and, every time a new EEG, report or whatever else has been identified in the source resource, a copy procedure starts to copy all available data to a target resource. Setting up the Microservice with the MHVT local resource as source and CloudVeneto local resource as destination, will ensure that data are automatically copied in background to the target resource. The result of this process is that patient with new exams is available in CloudVeneto resource, and in the end all exams are

scanned by Django server and persisted in local database, allowing patient's data management.

7.4.4 Online data visualization (EEG and Reports)

Another useful feature would be the capability to view EEG data and reports online, directly from the examination page. With this feature, users wouldn't need to download EEG and report data to view them in an external editor. Instead, they could conveniently visualize both the raw EEG data, cleaned data and word documents directly within their web browser. This would facilitate the process and enhance user experience by eliminating the need for additional software or steps.

7.4.5 Scalability to different worker machines

To minimize computational power in case of multiple analysis requested at the same time, it may be necessary to delegate the analysis algorithm execution to a pool of worker machines rather than one, to make sure that memory and power of virtual machines is correctly managed to get data back into server in the lowest time possible, increasing performance of the system. In this project, a single machine is responsible for data analysis and its parameters are hardcoded but in future, it may be interesting to exploit different machines for analysis purposes.

Bibliography

- [1] Millet D. THE ORIGINS OF EEG. 7th Annual Meeting of the International Society for the History of the Neurosciences (ISHN)
- [2] <https://www.emotiv.com/eeg-guide/>
- [3] The Wave - The characteristics of an EEG — Firstclass (firstclassmed.com)
- [4] Jacobs J, Staba R, Asano E, Otsubo H - High-frequency oscillations (HFOs) in clinical epilepsy. Prog Neurobiol. 2012
- [5] Oriano Mecarelli - Theoretical-Practical Manual of Electroencephalography
- [6] Nagel, Sebastian. (2019). Towards a home-use BCI: fast asynchronous control and robust non-control state detection
- [7] Trambaiolli, L.R., Lorena, A.C., Fraga, F.J., Kanda, P.A., Nitrini, R., & Anghinah, R. (2011). Does EEG Montage Influence Alzheimer's Disease Electroclinic Diagnosis? International Journal of Alzheimer's Disease, 2011.
- [8] <https://www.raosentcare.com/sleep-lab/>
- [9] Micromed Brain Quick Software Documentation
- [10] <https://www.ibm.com/topics/cloud-computing>
- [11] <https://www.practicallogix.com/services/cloud-engineering/>
- [12] <https://zeptosystems.com/new-technology-trends-for-2022/>
- [13] <https://userguide.cloudveneto.it/en/latest/>
- [14] S. Bertuccio, G. Tardiolo, F. M. Giambò, G. Giufrè, R. Muratore, C. Settimo, A. Rafa, S. Rigano, A. Bramanti, N. Muscarà and M. C. De Cola - ReportFlow: an application for EEG visualization and reporting using cloud platform, 2021
- [15] <https://www.emotiv.com/emotiv-eeg-cloud/>
- [16] Qi Tian, Wen Wu, Qin Zhu, Tao Cai - A Cloud-based Data Platform for Efficient EEG Data Management, Collaboration, and Analysis, 2023

- [17] Jezek P., Vareka L. - Cloud Infrastructure for Storing and Processing EEG and ERP Experimental Data, 2019
- [18] <https://docs.djangoproject.com/en/5.0/>
- [19] <https://learndjango.com/tutorials/what-django-python>
- [20] <https://www.django-rest-framework.org/>
- [21] <https://docs.celeryq.dev/en/stable/index.html>
- [22] <https://tiptopsecurity.com/how-does-https-work-rsa-encryption-explained/>
- [23] <https://micromedgroup.com/products/brainquick/brainquick-ambulatory/>

Acknowledgements

Con questo progetto si chiude anche un capitolo della mia vita; la mia esperienza lavorativa in Micromed, infatti, volgerà al termine poco dopo la conclusione del mio percorso di studi. Desidero dedicare un pensiero speciale a tutti coloro che mi hanno sostenuto in ambiente lavorativo, universitario e privato, rendendomi la persona che sono oggi.

Come prima cosa, vorrei esprimere la mia gratitudine a Micromed, un luogo che non solo mi ha fornito un ambiente professionale stimolante, ma ha anche contribuito alla mia crescita come persona. Riconosco il contributo di tante figure, tra cui Raffaele, Alberto, Nicola, Federico, Giulia e Alessandra, preziose per la loro costante guida e i consigli che mi hanno offerto durante lo sviluppo del mio progetto di tesi e, soprattutto, nel mio percorso lavorativo. Non potrei mai dimenticare anche tutti gli altri colleghi di Micromed (siete tantissimi perdonatemi), una famiglia che mi ha accolto con affetto e con cui ho condiviso momenti indimenticabili. Il legame che abbiamo formato in questi anni va al di là del lavoro, è un legame di amicizia che sono sicuro rimarrà tale nel corso del tempo. Mi mancheranno moltissimo le partite a ping-pong, gli aperitivi, le cene e i momenti passati assieme, tra disperazioni e soprattutto gioie. Auguro il meglio a tutti e spero di aver lasciato un piccolo pezzo di me a ognuno di voi.

Un ringraziamento speciale e profondo alla mia famiglia, il pilastro fondamentale che mi ha sostenuto incondizionatamente durante gli anni di studio. A mia madre Lucia e mio padre Giulio va il mio più sincero apprezzamento e a loro dedico tutto, poiché sono stati la mia costante fonte di incoraggiamento e sostegno in ogni fase della mia vita e crescita. Mi avete insegnato a guardare il mondo da due prospettive diverse, permettendomi di esplorare, sperimentare e crescere, così da comprendere appieno i miei veri desideri. Mai una parola fuori posto e mai avete forzato la mano per imporvi sulle mie scelte di vita, e per questo ve ne sarò eternamente grato. Vi ho sempre considerato dei modelli, sia nei momenti positivi che in quelli più difficili. Con il passare del tempo, mi rendo conto sempre più che le persone che, scherzando, mi dicevano “sei tutto tuo padre” avevano ragione. Ogni giorno, vedo sempre più di me stesso nei tuoi modi di fare e di relazionarti con gli altri, e ne vado fiero. Avrei voluto tanto che tu e nonno poteste essere qui accanto a me per celebrare questo momento, ma è anche vero che la vita riserva imprevisti e non gioca sempre a carte scoperte. Io credo che ogni esperienza, positiva o negativa che sia, vada vissuta come una sfida, per crescere e dare il meglio di sé stessi, sempre. Vorrei ringraziare anche tutti i miei zii, cugini, nonni e parenti per la fiducia che hanno sempre riposto in me. Un ringraziamento particolare va a Francesco, tanto zio quanto padre, per tutti i fantastici momenti passati assieme, sempre disponibile ad aiutarmi e discutere di qualsiasi cosa, dandomi il suo prezioso punto di vista e il suo supporto. Ultimo ma non per importanza, un immenso grazie a Enrico, più che cugino un fratello, con cui sono cresciuto e ho condiviso i momenti più belli della mia vita. So che ormai siamo grandi e ben presto arriverà il momento di separarci, ognuno per la sua strada, ma sappi che sono e sarò sempre orgoglioso di te, indipendentemente dalle scelte che farai, e spero che il nostro legame di ferro non svanisca con il passare del tempo.

Desidero esprimere un profondo ringraziamento a tutti miei amici, compagni di squadra e di allenamento, che hanno giocato un ruolo chiave nel mio percorso di

vita. Le vostre parole di incoraggiamento, il sostegno costante e la vostra presenza hanno reso il viaggio fino a questo momento ancora più significativo. Grazie per le lunghe serate trascorse a discutere idee, per le risate condivise nei momenti di stress e per le avventure che abbiamo vissuto insieme. Il vostro sostegno mi ha ispirato e motivato a dare il meglio di me. Che il nostro legame di amicizia rimanga saldo nel tempo, pronto a celebrare futuri successi e a sostenerci reciprocamente nelle sfide che ci attendono. Ciascuno di voi mi ha donato un pezzo di sé e ha contribuito a rendermi la persona che sono oggi, e per questo vi ringrazio di cuore. Un grazie anche alle persone che hanno significato tanto nella mia vita e con cui ho condiviso anni di risate, affetto, esperienze e viaggi, che purtroppo oggi non sono più al mio fianco.

Vorrei ringraziare anche tutti i miei colleghi dell'università con cui ho collaborato, dal primo all'ultimo, in questi cinque anni. Abbiamo saputo ridere, scherzare, disperare e sostenerci a vicenda, con innumerevoli giornate passate in aula studio a fare esercizi, concludere progetti, rivedere appunti. Grazie per avermi sempre motivato e spinto a dare il meglio di me, siete stata una vera fonte di ispirazione e spero che ognuno di voi trovi la sua strada e si senta realizzato.

Infine, un sentito ringraziamento va a Marco Castellaro e Giovanni Sparacino, i professori che hanno dato il via a questo progetto e mi hanno aiutato a portarlo a compimento indirizzandomi e spronandomi a fare sempre meglio, e a tutte le persone che hanno contribuito fornendo informazioni preziose e condividendo la loro esperienza.

Grazie di cuore a tutti, ognuno ha giocato un ruolo chiave nel mio percorso in questi anni, e difficilmente dimenticherò ciò che ognuno ha significato per me.

Un saluto speciale,

Antonio

P.S.: tutto troppo sentimentale

