Alma Mater Studiorum · Università di Bologna

Scuola di scienze
Corso di Laurea in Ingegneria e Scienze Informatiche

SVILUPPO DI UN APPLICATIVO WEB REAL-TIME ORIENTATO AL BENESSERE COGNITIVO DI UN TEAM

Elaborato in
INGEGNERIA DEL SOFTWARE

Relatore:
Prof. ALESSANDRO RICCI
Co-relatori:
Dott. LUCA TALEVI
Dott. MARCO RENZI

Presentato da:
ALESSANDRO TALMI

Seconda Sessione di Laurea
Anno accademico 2019-2020

Indice

Introduzione

Nel corso degli anni sono stati compiuti passi da gigante nel campo delle neuroscienze, portando nel mondo reale idee che trovavano realizzazione soltanto nei romanzi di fantascienza.

Seppur la strada sia ancora lunga, oggi siamo sempre più vicini ad avere una comprensione almeno parziale dei fenomeni che governano la nostra mente.
Tali conoscenze, se sfruttate nel modo corretto, potrebbero portare a migliorare la condizione di vita di molte persone.

Ma non sono soltanto le neuroscienze ad aver portato a termine grandi progressi; come ormai accade in gran parte delle discipline, l'informatica gioca un ruolo di supporto sempre più preponderante, fornendo nuove tecnologie ad uso e servizio dei campi scientifici ed umanistici più disparati.
È quando neuroscienze ed informatica collaborano che ciò che sembrava impossibile trova un'opportunità per divenire reale.

Non è raro che assottigliare il confine tra l'uomo e le macchine possa provocare un'iniziale paura nelle persone; il timore è che tali tecnologie ci possano allontanare dalla nostra natura umana portandoci a divenire esseri di natura artificiale.
Seppur queste domande sorgano spontanee, ciò a cui bisogna guardare in realtà è l'obbiettivo con la quale una nuova tecnologia viene concepita.

Ciò che si sta cercando di portare a termine non è la castrazione della nostra umanità, bensì l'ampliamento di essa aiutandola dove spesso i nostri limiti umani ci portano a provare disagi e sofferenze.
Dopotutto anche un paio di occhiali da vista sono per definizione artificiali, ma non ci rendono meno umani.
Attraverso di essi anzi possiamo godere a pieno della nostra vista sebbene il corso naturale degli eventi ci avrebbe relegato ad una vita dai toni sfocati.

L'obbiettivo ultimo di questa tesi è la progettazione e lo sviluppo di un applicativo in grado di aiutare i lavoratori a soffrire il meno possibile la stanchezza mentale, migliorando il loro umore e, nel lungo andare, la loro salute.
Un effetto secondario al benessere della psiche di un individuo sul lavoro, è l'incremento della produttività in quest'ultimo.

Si viene dunque a creare una situazione in cui vincono tutti: da un lato i lavoratori sono più felici e rilassati e, dall'altro, l'azienda ne guadagna in flusso produttivo.
Non ci sono dunque scuse, è impossibile evitare di considerare il valore che una tale soluzione porterebbe nelle vite dei lavoratori e delle aziende.

In questa tesi dunque seguiremo lo sviluppo di tale soluzione dal punto di vista dell'ingegneria del software.
Ci muoveremo dall'alto verso il basso; una volta compreso a pieno il contesto, il sistema verrà modellato e progettato nella sua interezza così da poterlo implementare scegliendo le soluzioni tecnologiche più adeguate.

Capitolo 1

Contesto

Questo capitolo si prepone di esplicare il contesto nel quale opera questo progetto di tesi.
Inizieremo esplicando dei concetti nell'ambito delle Neuroscienze: *knowledge-workers e carico cognitivo*.
Successivamente vedremo come quanto esplicato si vada ad intersecare con un contesto informatico attraverso le *interfacce neurali per il monitoraggio dello stato mentale*.
Infine, vedremo come *Vibre*, l'azienda ospitante, si inserisca in questo ambito tramite *la piattaforma Atlas e l'algoritmo Mindpulse* da loro sviluppato, componente integrante del macro progetto in cui si inserisce questa tesi: *NeuroFrame*.

1.1 Knowledge-workers e carico cognitivo

Secondo una definizione antecedente al 2007 i "lavoratori della conoscenza" (**knowledge workers**) sono coloro che "manipolano direttamente dei simboli per creare un prodotto di conoscenza originale o per fornire valore aggiunto ad un prodotto già esistente".
Il termine "knowledge workers" fu utilizzato per la prima volta nel 1959 da Peter Drucker nel libro "Landmarks of Tomorrow".
In senso più ampio i Knowledge workers sono tutti coloro che trasformano la propria conoscenza professionale e i propri input conoscitivi in output di conoscenza con un valore aggiunto rispetto a quello di partenza [1].
Questo tipo di lavoro consuma dunque principalmente risorse cognitive piuttosto che fisiche; suddette risorse inoltre vengono consumate non solo in contesto lavorativo, ma da praticamente ogni attività che viene svolta nel corso di una giornata.
Dipendentemente dal tipo di attività svolta e da quanta familiarità si ha con essa la quantità di risorse mentali impiegata varia significativamente.
Definiamo dunque il consumo di risorse mentali per una determinata attività come **carico cognitivo**.
Studi in materia evidenziano inoltre come l'incidenza del carico cognitivo e le performance

di un individuo siano collegate anche all'affaticamento accumulato nel corso del tempo; un individuo che esegue un'attività in una condizione di stress otterrà risultati peggiori ed impiegando un maggiore sforzo rispetto allo stesso individuo che provenga da una situazione di relax.
Questo fenomeno viene chiamato "affaticamento mentale" (**Mental fatigue**) [2].
Distinguiamo due tipi di Mental Fatigue:

- *Acute Mental Fatigue*
  Questo tipo di affaticamento riguarda una circostanza con durata nel breve periodo; questa condizione può essere superata da un individuo attraverso del semplice riposo.

- *Chronic Mental Fatigue*
  Si parla di affaticamento mentale cronico quando una condizione di stress per l'individuo viene protratta nel lungo periodo ed il riposo non è sufficiente a superare questa condizione. La permanenza di un individuo in questo stato aumenta drasticamente l'incidenza di un fenomeno chiamato **Occupational Burnout** [3], la condizione in cui un individuo supera il proprio limite cognitivo con gravi conseguenze emotive, comportamentali e fisiche.

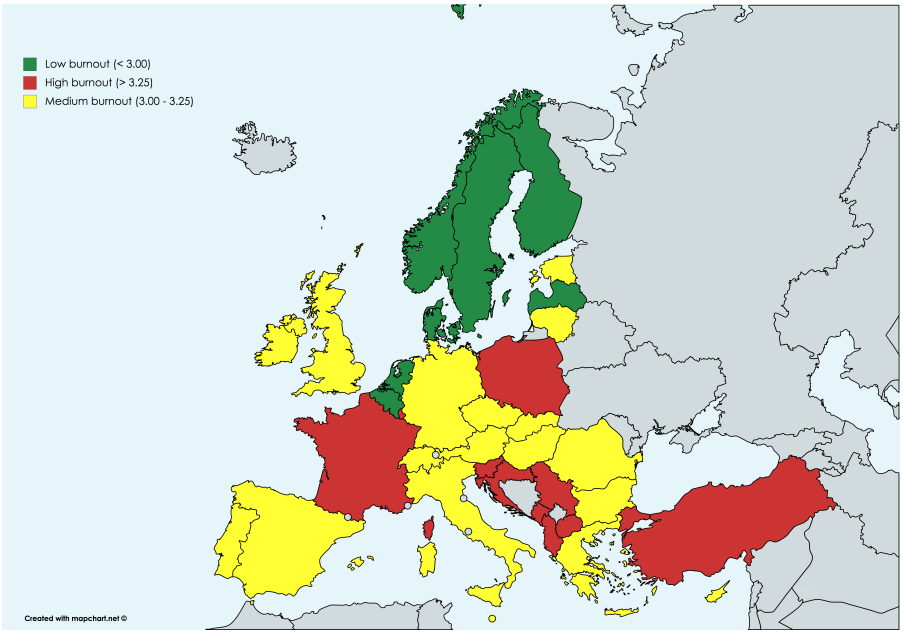

Figura 1.1: Livello medio di Burnout in Europa(2015)[scala da 1 a 5] [4]

La gestione delle risorse mentali nell'era digitale dunque diventa un aspetto fondamentale per la salute dei lavoratori ed in generale di tutti gli individui.
Una corretta gestione delle risorse mentali inoltre può incrementare drasticamente la qualità del lavoro prodotto, permettendo ad un individuo di raggiungere lo stato di **Flow** [5]; questo stato si ottiene quando si dedica completa attenzione ad un'attività di cui si è appassionati ed in cui si è totalmente immersi.
Durante il Flow le normali sensazioni che distraggono un individuo dal lavoro (come affaticamento, fame, stimoli fisici ed ambientali) scompaiono permettendo la massima resa nel lavoro impiegando il minimo sforzo.

1.2 Interfacce neurali per il monitoraggio dello stato mentale

Un' interfaccia neurale, nota anche con il termine inglese **Brain-computer interface** (*BCI*), è un mezzo di comunicazione diretto tra un cervello (o più in generale parti funzionali del sistema nervoso centrale) ed un dispositivo esterno quale ad esempio un computer [6].
Il dispositivo campiona l'attività cerebrale attraverso tecnologie quali l'elettroencefalogramma (EEG), la spettroscopia funzionale nel vicino infrarosso (fNIRS), ecc...
L'attività cerebrale campionata viene quindi codificata in una forma comprensibile dal calcolatore.
A questo punto i dati ottenuti vengono elaborati da algoritmi il quale scopo è trarre delle metriche che esprimano in modo comprensibile un aspetto cognitivo ricercato (come ad esempio il carico cognitivo visto nella sezione precedente).

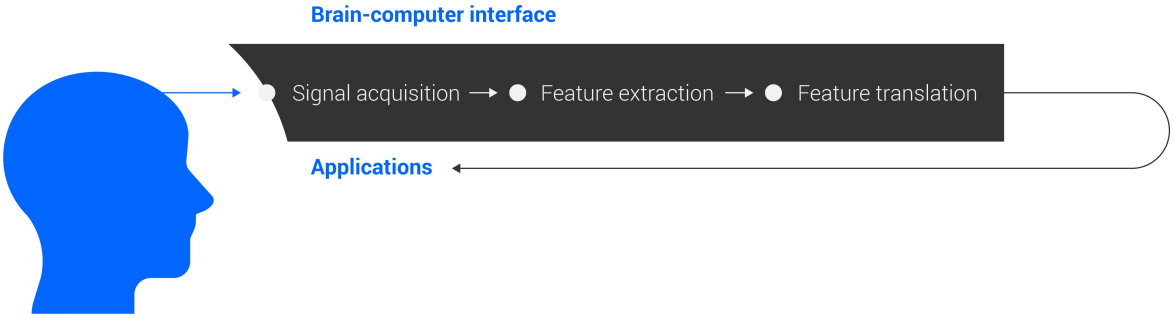

Figura 1.2: Funzionamento ad alto livello di una BCI

Il primo obiettivo che portò alla concezione e realizzazione di BCI fu fornire a persone con disabilità canali di comunicazione o controllo di dispositivi esterni.

Sistemi di questo tipo però rimangono confinati in un contesto medico per due importanti motivazioni:

- *Alta specificità delle feature ricercate*
- *Hardware costoso ed ingombrante*

Nell'ultimo decennio i ricercatori hanno cominciato a prendere in considerazione l'utilizzo delle BCI in contesti che coinvolgano soggetti sani.
Il significato del termine BCI si è dunque evoluto nel tempo; inizialmente il termine si riferiva solamente alla traduzione dell'intento di un utente per attuare una risposta nel mondo fisico.

Oggi invece il termine si è espanso fino a comprendere il monitoraggio dello stato mentale ed emotivo di un individuo. Questo nuovo tipo di BCI è stato denominato **passive Brain-computer interface** (*pBCI*) [7].
Negli ultimi anni le pBCI sono migliorate incredibilmente in termini di affidabilità, usabilità e contesti di applicazione.
Sono state pubblicate grandi quantità di ricerche riguardanti il potenziale delle pBCI; tuttavia, la maggior parte dei lavori e test è stata condotta in ambiente di laboratorio o comunque ambienti altamente controllati.

1.3 La piattaforma Atlas e l'algoritmo Mindpulse

Vibre, un'azienda nel territorio cesenate, si prepone come mission e come modello di business l'idea di rendere facilmente fruibile ad un vasto pubblico il valore informativo che le neuroscienze hanno da offrire attraverso delle interfacce neurali passive, facili da usare e facilmente scalabili.
Per ottenere questo risultato Vibre ha lavorato negli anni alla piattaforma **Atlas**, un *servizio cloud-based preposto all'elaborazione di segnali EEG* (ottenuti attraverso i dispositivi *Muse* [8] (*figura 1.3*)) al fine di ottenere delle metriche che descrivano lo stato cognitivo di un soggetto.

Figura 1.3: Dispositivo Muse2 [9]

Atlas (*figura 1.4*) è descrivibile ad alto livello come un sistema composto da:

- **Algorithm packages**
  L'insieme dei tre algoritmi Python sviluppati da Vibre capaci di estrapolare metriche cognitive elaborando segnali EEG:
  - *MindFeel*
  - *MindPulse*
  - *MindPrint*
- **Neuroserver**
  Un server REST che espone gli endpoint dei tre algoritmi ai servizi che ne richiedono la computazione.

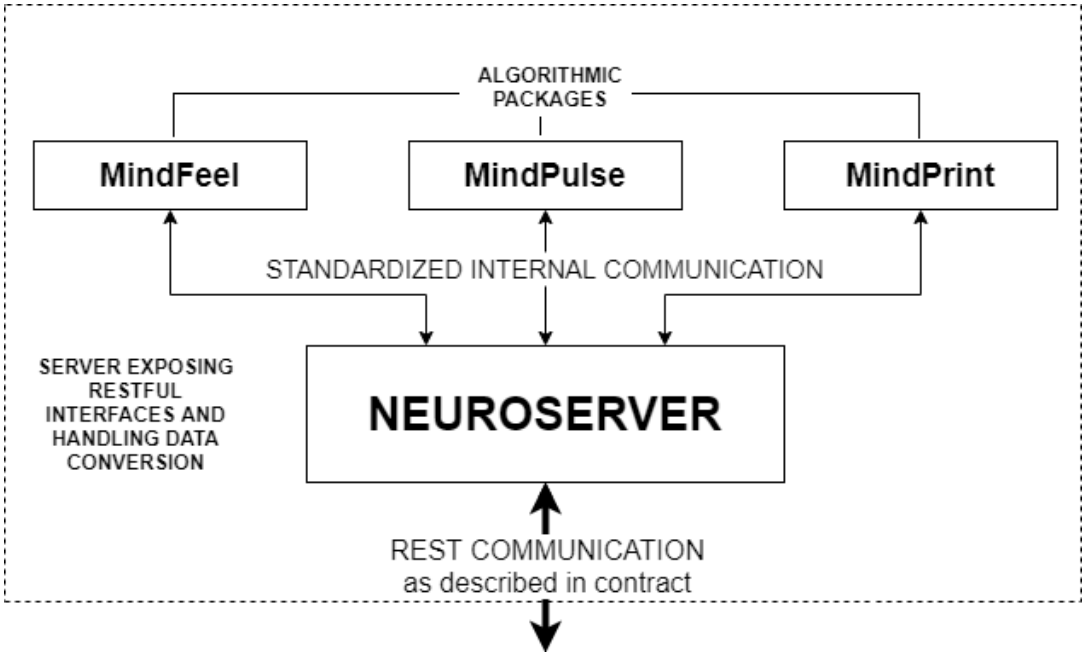

Figura 1.4: Vista ad alto livello della piattaforma Atlas

Un servizio erogato ad un'azienda dunque, ottenendo dati dal dispositivo Muse, invierà i segnali EEG a Neuroserver che si occuperà di consegnarli all'algoritmo giusto al fine di ottenere dati rilevanti per il servizio in uso.

Ai fini di questa tesi discuteremo brevemente e ad alto livello l'algoritmo **MindPulse**, così da meglio comprendere il progetto svolto.
MindPulse, attraverso della variazione delle onde Alpha, Beta, Theta e Delta emesse dal cervello e rilevate dal dispositivo Muse, è in grado di determinare delle metriche di facile interpretazione che descrivono il benessere cognitivo di un individuo.
In modo particolare l'algoritmo (dopo aver effettuato una calibrazione sul soggetto) permette di comprendere:

- *Carico cognitivo (sezione 1.1)*
  Descrive il carico di lavoro compiuto al momento da un individuo, andando da uno stato di massimo relax fino ad uno stato di massimo impegno.

- *Affaticamento mentale (sezione 1.1)*
  Descrive l'affaticamento mentale accumulato nel tempo (ad esempio dopo una intensa mattinata di lavoro un individuo risulterà maggiormente affaticato ed avrà a disposizione meno "risorse mentali").

Questo algoritmo è dunque alla base del *m*acro-progetto all'interno del quale si inserisce questa tesi: **NeuroFrame**.

1.4 Neuroframe

È normale, per un qualsiasi sportivo o lavoratore manuale, *tenere sotto controllo il proprio livello di fatica fisica durante la giornata e prendere decisioni basandosi su di essa*; il lavoratore potrebbe decidere ad esempio se fare una pausa, quanta energia conservare per quel fondamentale scatto finale, se sollevare o no quel peso, se fare stretching...
La sensoristica corporea che permette al nostro cervello di compiere queste scelte è numerosa, precisa e generalmente efficiente.

Lo stesso discorso purtroppo non vale per tutti i cosiddetti Knowledge-workers (*sezione 1.1*): i programmatori, i ricercatori, gli architetti, tutti coloro il cui lavoro richiede uno sforzo principalmente mentale piuttosto che fisico.
Il cervello dovrebbe dunque monitorare il proprio funzionamento, ma non può farlo con efficacia: *gli stessi sensori che dovrebbero controllare l'affaticamento vengono affaticati dallo sforzo!*

Per questo motivo, i Knowledge-workers usano spesso strumenti esterni per controllare i propri sforzi, primi tra tutti orologi e tecniche di time-boxing (es. Pomodoro Timer).
Il continuo crescere di casi di Occupational Burnout (*sezione 1.1*) suggerisce che questi strumenti non bastano; è necessario fornire al cervello un equivalente, anche approssimativo, di quella sensoristica che esso è in grado di usare per le attività fisiche: una metrica per il carico cognitivo.

Per risolvere questo problema è nato *NeuroFrame*.
Attraverso i modelli per il carico cognitivo e per l'affaticamento mentale di MindPulse (*sezione 1.3*), NeuroFrame si propone come una *suite Software as a Service* capace di monitorare lo stato di un numero arbitrario di persone (team) in real-time, con **dashboard** capace di mostrare le metriche di ogni componente del team ed **applicativi di acquisizione di segnali EEG** per ciascun individuo.

Sulla base delle metriche mostrate tramite dashboard un *Team manager* potrà prendere decisioni mirate ad incrementare il benessere cognitivo del team.
NeuroFrame offrirà capacità di:

- *Prevenire i burnout*
- *Valutare il carico cognitivo*
- *Misurare l'affaticamento mentale*

Vibre si prepone dunque di portare questo servizio nelle aziende attraverso un prodotto scalabile e semplice da usare per l'utente finale, così da migliorare la produttività ed il benessere dei lavoratori nelle aziende.
L'impresa non è tra le più semplici in quanto, come affrontato nella *sezione 1.1*, il campo delle pBCI offre grandi potenzialità ma manca di un vasto parco servizi rivolto ad ambienti che esulano dallo sperimentale o dal medico.

Capitolo 2

Studio del problema

In questo capitolo verrà affrontato il problema nella sua globalità, fornendo una comprensione a tutto tondo del sistema NeuroFrame.
In primo luogo si effettuerà un'*analisi dei requisiti del sistema*, esplicando il criterio con il quale i requisiti sono stati scelti, oltre che i requisiti stessi.
Seguirà una *modellazione dei requisiti del sistema* e, attraverso numerose tecniche fornite dall'ingegneria del software, verrà modellato in toto il comportamento del sistema.
Successivamente verranno presentati i *wireframe* degli applicativi, naturale conseguenza delle fasi precedenti ed essenziali per guidare il processo di sviluppo.
Da quanto trattato poi emergeranno degli *aspetti critici relativi all'applicativo Dashboard*, dettati da quanto emerso negli step subito antecedenti.

2.1 Analisi dei requisiti del sistema

Analizziamo ora quali requisiti il sistema deve soddisfare.
L'approccio scelto mette al centro le **user stories**, cioè funzionalità che l'utente finale deve poter trovare nel prodotto.
Ogni user story quindi è identificabile come un **task** da completare.

Non tutti i task però hanno pari importanza, specialmente nell'ottica della prototipazione e della graduale crescita delle feature del prodotto.
Occorre quindi stabilire quali siano le feature base ricercate alla prima release del prodotto ma, contemporaneamente, tenere a mente feature future per creare un sistema già predisposto ad implementarle.
Per questa ragione è stato scelto il metodo di prioritizzazione **MoSCoW**.

2.1.1 Prioritizzazione tramite metodo MoSCoW

La metodologia MoSCoW è una tecnica di **prioritizzazione** molto usata nell'analisi di sistemi informatici.

Le lettere maiuscole dell'acronimo simboleggiano le 4 categorie fornite da questo metodo per la suddivisione della priorità e il conseguente impiego di risorse umane, economiche e temporali:

- **Must have**
- **Should have**
- **Could have**
- **Won't have** (o, alternativamente, *Wish*)

Questa metodologia è stata creata dallo sviluppatore software *Dai Clegg* e, successivamente, donata al Dynamic Systems Development Method (*DSDM*).
Inizialmente MoSCow fu concepito come un framework di prioritizzazione per sviluppo di applicazioni in tempi brevi; oggi tuttavia, grazie alla popolarità acquisita, il metodo è stato ampliato per gestire varie tipologie di progetto [10].

2.1.2 Must have

Attraverso i Must have viene identificato il **Minimum Usable Subset** (*MUS*), cioè le feature minime che il progetto deve obbligatoriamente soddisfare per essere considerato fruibile.
È qui che vengono specificati i task con la **massima priorità** e dove vengono spesi, almeno inizialmente, maggiore tempo e risorse.

Per il sistema NeuroFrame sono stati identificati i seguenti must have:

- *Monitoraggio real-time del carico cognitivo di ogni persona*
  Dev'essere possibile valutare in tempo reale il carico cognitivo di ogni singolo componente del team.
- *Dashboard del team dove il team manager può osservare l'andamento di tutti i monitorati*
  Deve esistere una sezione della Dashboard che esprima facilmente lo stato cognitivo dell'intero team.

- *Ogni utente ha la propria applicazione che usa in autonomia per far partire la registrazione*
  La possibilità di un utente all'interno di un team di inviare dati dev'essere totalmente indipendente dallo stato dei team manager o degli altri componenti del team.

- *Signup/login integrato per gli utenti monitorati*
  Ogni utente monitorato facente parte di un team possiede le proprie credenziali aziendali attraverso le quali effettuare il login nella propria istanza dell'applicativo di registrazione.

- *Ogni utente deve essere sottoposto ad una fase di calibrazione*
  Al fine di ottenere delle metriche significative per l'individuo il dispositivo di calibrazione necessita di una fase di calibrazione.

- *Il team manager deve poter effettuare un login sulla dashboard*
  Ogni team manager deve avere le proprie credenziali aziendali attraverso le quali effettuare login al fine di poter accedere ai vari team presenti in azienda.

2.1.3 Should have

I task definiti come Should have ricoprono molta importanza all'interno di un progetto ma **non sono essenziali al funzionamento del sistema** e al rilascio di una prima versione; hanno dunque una priorità inferiore rispetto ad i Must have.
È dunque qui che vengono specificati **miglioramenti delle performance**, **miglioramenti strutturali** o **feature future**.

I Should have identificati sono i seguenti:

- *Storico degli andamenti*
  La possibilità per i team manager di visionare i dati cognitivi di registrazioni passate dei membri di un team.

- *Tutte le applicazioni dovrebbero essere web app*
  Implementare gli applicativi utilizzati da utenti e team manager al fine di aumentare la scalabilità del prodotto ed ottenere la massima compatibilità con il maggior numero possibile di dispositivi.

- *Possibilità di gestione on-premise*
  Alternativamente a quanto specificato del task precedente, si vuole offrire la possibilità di utilizzare il servizio con applicativi installati nelle macchine del cliente.

2.1.4 Could have

Scendendo ulteriormente nel grado di priorità troviamo gli Should have.
Essi non sono necessari al funzionamento del sistema ma, rispetto agli Should have, **hanno un impatto notevolmente minore nelle funzionalità offerte dal prodotto finito**.
Generalmente nella creazione di un prodotto sono i primi ad essere scartati e/o rimandati nel caso Must have e Should have si rivelino avere costi e tempistiche maggiori rispetto a quanto preventivato.

Sono stati individuati due task appartenenti a questa categoria:

- *Possibilità di centralizzare le registrazioni per il manager*
  Consentire ad un team manager una forma di controllo sulle registrazioni dei membri del team (ad esempio fermandole o facendole partire).

- *Possibilità di esportare gli storici*
  Espandendo il task riguardante la possibilità di visionare lo storico degli andamenti si vuole offrire al cliente la capacità di esportare tali storici.

2.1.5 Won't have

Troviamo infine i task definiti come Won't have o, come accennato alla *sezione 2.1.1*, i Wish.
Ciò che viene posto in questa categoria detiene il **grado di priorità più basso possibile** e rappresenta idee emerse in corso d'opera sulla cui fattibilità e valore verrà discusso in futuro.

NeuroFrame propone questi Wish:

- *Dashboard del singolo utente*
  La possibilità di avere una dashboard dedicata all'utilizzo di "team" composti solo da una persona.

- *Andamento real-time sull'applicazione dell'utente*
  Offrire la possibilità al membro di un team di visualizzare il proprio benessere cognitivo all'interno della propria istanza dell'applicazione di registrazione.

- *Test psicometrici di affaticamento e stato di flow, confrontati in maniera automatica con le misurazioni*
  La prospettiva di includere test psicometrici volti a valutare la propensione all'affaticamento di un individuo e la sua capacità di raggiungere lo stato di flow (*sezione 1.1*).

2.2 Modellazione dei requisiti del sistema

Partendo da quanto detto fin ora è giunto il momento di modellare i requisiti del sistema.

Al fine di ottenere una resa quanto più possibile chiara e completa si è scelto di *modellare il sistema NeuroFrame da cinque differenti punti di vista*; in particolare, nel tentativo di rendere la fruizione il più semplificata possibile, *i modelli verranno presentati partendo da aspetti maggiormente globali fino a raggiungere comportamenti del sistema più dettagliati.*

I modelli scelti sono i seguenti:

- *Casi d'uso*
  Come naturale conseguenza della *sezione 2.1* saranno formalizzati i casi d'uso del sistema.
- *Sottosistemi coinvolti*
  Il macrosistema NeuroFrame è a sua volta composto da sottosistemi indipendenti che comunicano tra loro; qui dunque saranno presentati tali sottosistemi.
- *Flusso informativo*
  Compreso quali siano i sottosistemi che compongono NeuroFrame verrà dunque esplicato il flusso informativo che intercorre tra di essi.
- *Analisi orientata agli oggetti*
  Qui saranno esplicate ad alto livello le entità che caratterizzano il problema.
- *Analisi orientata agli stati*
  Infine verranno esplicati i modelli di più difficile comprensione, rappresentanti gli stati che i sottosistemi possono assumere.

2.2.1 Casi d'uso

Cominciamo modellando i casi d'uso del sistema nella sua interezza (*figura 2.1*).

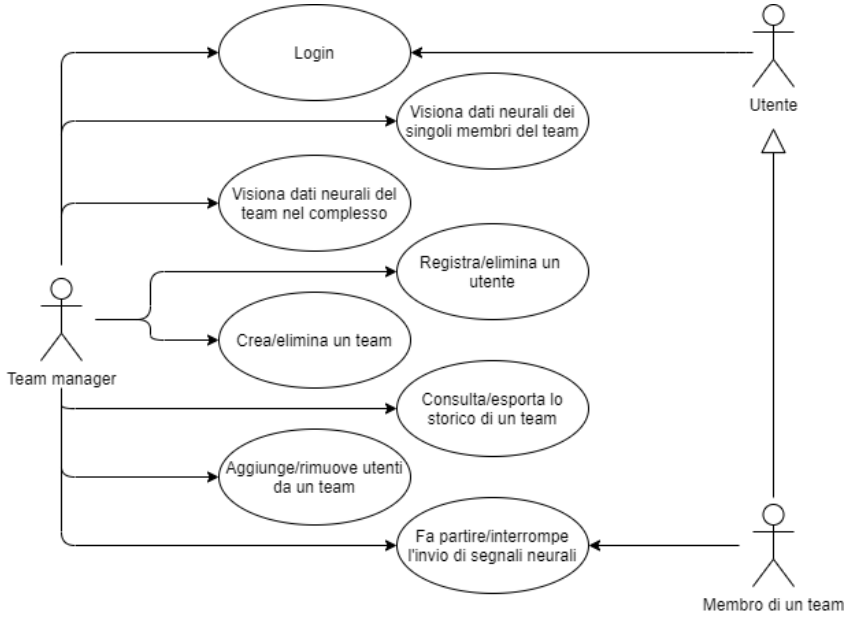

Figura 2.1: Casi d'uso del sistema NeuroFrame

Troviamo due attori base: il **Team manager** e l'**Utente**; quest'ultimo, previa azione del Team manager, può specializzarsi nel **Membro di un Team**.
Analogamente il Membro di un Team può tornare ad essere un Utente base grazie all'intervento del Team manager.

La possibilità di far iniziare e di interrompere lo streaming dei segnali neurali, così come il Login, è un caso d'uso comune a tutti gli attori.

Le restanti casistiche sono interamente competenza del Team manager e possiamo distinguerle in due categorie:

- *Gestionali*
  Egli si occuperà di creare/eliminare Utenti, aggiungere/rimuovere Utenti da un Team e crea/elimina i Team stessi.

- *Dati*
  Potrà visionale il benessere cognitivo dei singoli individui così come il benessere del Team nel complesso; tali dati potranno anche essere esportati.

2.2.2 Sottosistemi coinvolti

Vediamo ora i sottosistemi che compongono il macrosistema NeuroFrame (*figura 2.2*).

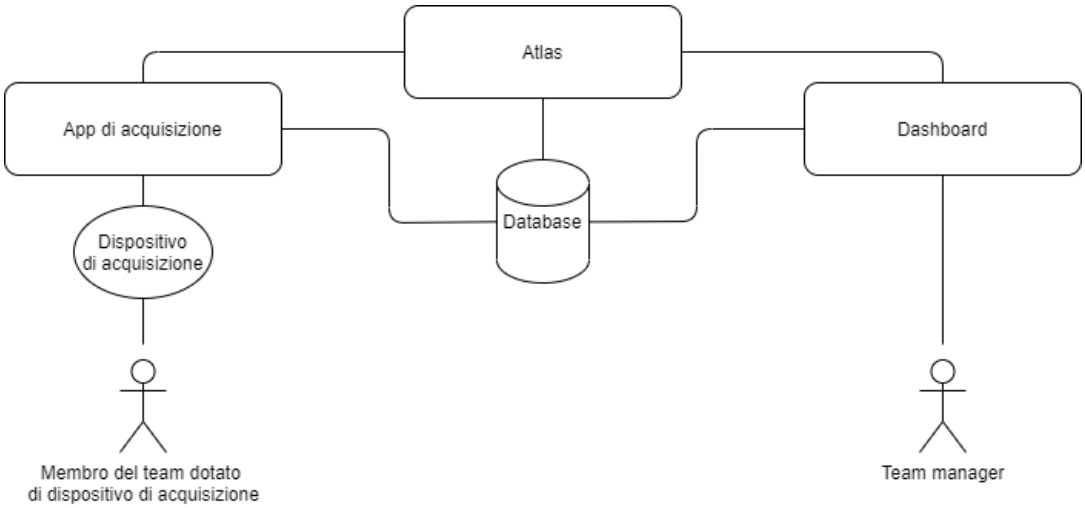

Figura 2.2: Sottosistemi che compongono il macrosistema NeuroFrame

La **piattaforma Atlas** (introdotta nella *sezione 1.3*), si presenta come lo snodo centrale del macrosistema; essa riceve dati sullo stato cognitivo di un individuo e, dopo averli elaborati attraverso l'algoritmo che dà il nome all'intero progetto, li memorizza in un **database**.

Il compito di inviare i dati riguardo lo stato cognitivo viene svolto dall'**App di acquisizione**. Ogni **membro di un team** utilizza un'istanza di tale applicativo e, attraverso un **Dispositivo di acquisizione**, ottiene passivamente un campionamento sul suo stato cognitivo. L'App di acquisizione inoltre comunica con il database al fine di autenticare un individuo attraverso le sue credenziali.

Infine la **Dashboard** si occupa di mostrare i dati elaborati dalla piattaforma Atlas ad un **Team manager**; egli monitorerà le condizioni di un team ed avrà accesso a tutte le opzioni che gli permetteranno di gestire il team. La Dashboard comunica con il Database al fine di recuperare i dati elaborati (oltre che per autenticare i Team manager). La comunicazione con la piattaforma invece esiste al fine di poter verificare lo stato attuale di un membro del team (es: online o offline).

Come si può notare dalla *figura 2.2*, i due endpoint utilizzati dal cliente (App di acquisizione e Dashboard) non sono in comunicazione diretta tra loro, ma utilizzano Atlas ed il Database come intermediari.

2.2.3 Flusso informativo

Partendo dall'ultima osservazione, è opportuno dividere il flusso informativo in due diversi diagrammi così da facilitarne la comprensione.

Il **primo diagramma** (*figura 2.3*) coinvolge il lato sinistro della *figura 2.2*, ponendo il focus sull'**App di acquisizione**.

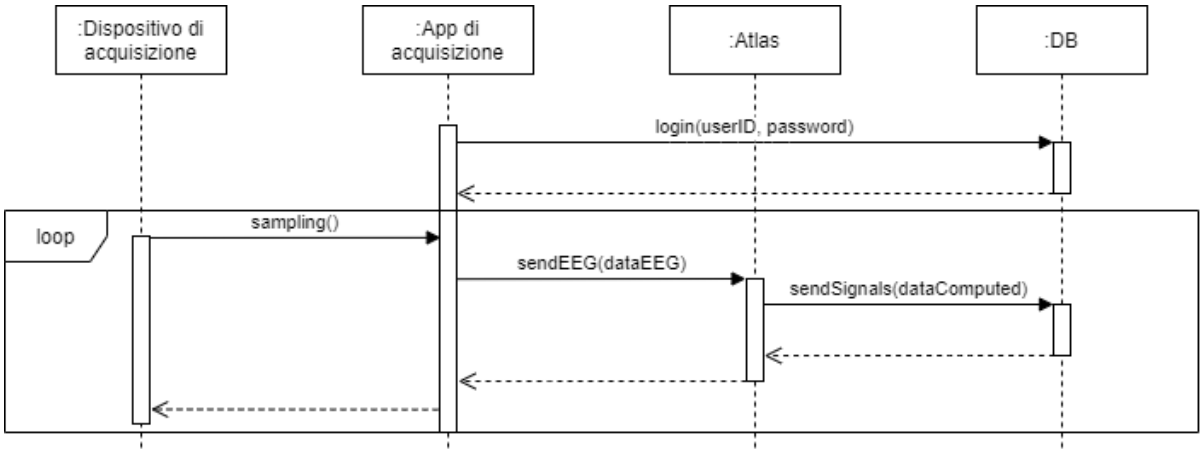

Figura 2.3: Diagramma di sequenza rappresentante il flusso informativo principale - lato applicativo di acquisizione

L'app di acquisizione effettua interrogazioni sul Database al fine di autenticare il membro del team.

Il processo che porta ad ottenere un dato neurale elaborato comincia invece dal Dispositivo di acquisizione che, periodicamente, campiona lo stato del membro del team ed invia tale informazione all'App di acquisizione; essa a sua volta trasmette tale dato alla piattaforma Atlas, che si occuperà di estrapolare metriche rilevanti da tale dato.
Al fine di rendere reperibili le metriche ottenute come ultimo passaggio Atlas salverà i risultati sul Database.

Tale azione di campionamento verrà ripetuta a cadenza periodica.

Il **secondo diagramma** (*figura 2.4*) coinvolge invece il lato destro dell'analisi dei sottosistemi coinvolti (*sezione 2.2.2*), chiarendo il flusso informativo che coinvolge la **Dashboard**.

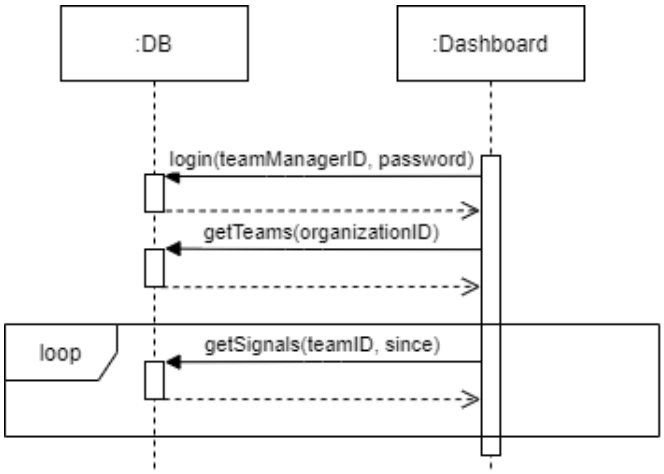

Figura 2.4: Diagramma di sequenza rappresentante il flusso informativo principale - lato Dashboard

Ottenute le credenziali del Team manager, la Dashboard interroga il Database al fine di identificarlo; nel caso ciò vada a buon fine la Dashboard esegue una nuova interrogazione al fine di ottenere tutti i team esistenti nell'organizzazione.

Una volta che il Team manager ha effettuato la scelta riguardo quale team monitorare, comincerà un'interrogazione periodica del Database da parte della Dashboard al fine di ottenere, se presenti, i nuovi dati elaborati appartenenti ad ogni membro del team.

La richiesta di nuovi segnali viene eseguita dalla Dashboard a cadenza periodica.

2.2.4 Analisi orientata agli oggetti

Osservando il sistema dalla prospettiva di un'analisi degli oggetti ad alto livello possiamo visionare le entità nel diagramma in *figura 2.5*.

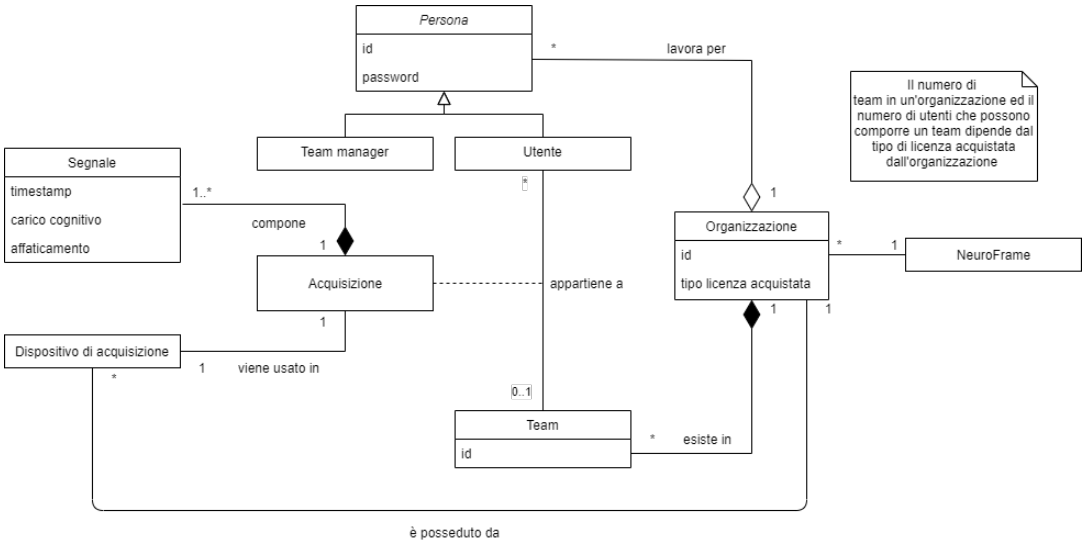

Figura 2.5: Analisi orientata agli oggetti del sistema NeuroFrame

All'interno del sistema **NeuroFrame** troviamo più **Organizzazioni**, distinte dal loro identificativo.

Per ogni Organizzazione abbiamo dei **Team** e delle **Persone**, che si specializzano in due diverse tipologie: **Team manager** e membro del team (chiamato per semplicità **Utente**); Persone e Team, così come le Organizzazioni, dispongono di un identificativo univoco.

Ogni Utente può appartenere ad un solo Team per volta; il numero di Utenti appartenenti ad un Team ed il numero stesso di Team che possono esistere all'interno di un'Organizzazione è vincolato al *tipo di licenza acquistata* da quest'ultima.

Un Utente che appartiene ad un Team può cominciare un'**Acquisizione**.
Ogni Acquisizione è composta da uno o più segnali (contenti le metriche calcolate da Atlas ed il timestamp corrispondente all'istante temporale in cui è avvenuto il campionamento) ed utilizza necessariamente un solo **Dispositivo di acquisizione** (che è posseduto da un'organizzazione).

2.2.5 Analisi orientata agli stati

Arriviamo dunque all'ultima analisi ed anche la più complessa.
Qui è stato deciso di modellare indipendentemente gli stati di App di acquisizione e Dashboard, a seguito del requisito di indipendenza funzionale tra i due sistemi (definito nella *sezione 2.1.2*).

Per facilitare la lettura sono state colorate in verde le strade che conducono al flusso d'esecuzione corretto e di rosso, viceversa, a quello scorretto; inoltre le azioni periodiche sono state evidenziate in blu.

Esplichiamo prima il comportamento dell'**App di acquisizione** (*figura 2.6*).

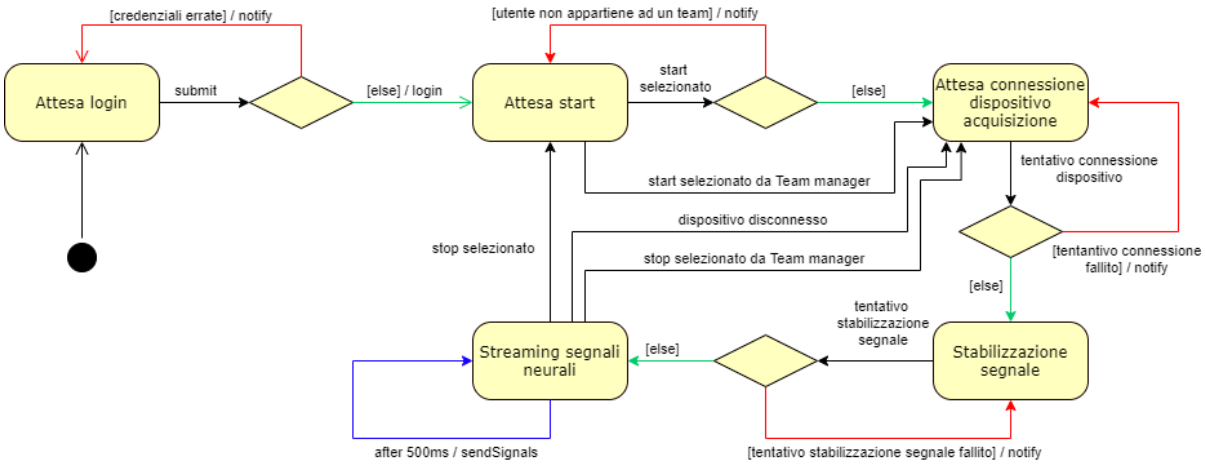

Figura 2.6: Analisi orientata agli stati - lato Applicativo d'acquisizione

Una volta effettuato il login il sistema attende l'input dell'Utente per procedere; un vincolo per transitare verso lo stato successivo però è rappresentato dall'appartenenza o meno ad un Team (come modellato nell'analisi orientata agli oggetti).
Alternativamente è possibile transitare verso il prossimo stato attraverso input remoto del Team manager.
Si passa dunque prima ad una connessione al Dispositivo di acquisizione poi, previa connessione eseguita, ad una stabilizzazione del segnale.

Una volta giunti allo stato in cui il sistema è correttamente in opera, l'app invia periodicamente (ogni 500 ms, al momento) alla piattaforma Atlas i dati ricevuti dal dispositivo di acquisizione in quel lasso temporale.

Dallo stato di streaming dei segnali neurali si può uscire per tre ragioni:

- L'Utente interrompe volontariamente lo streaming
- Il Dispositivo di acquisizione si disconnette
- Via remoto il Team manager interrompe lo streaming dell'Utente

Modelliamo ora gli stati dell'applicativo **Dashboard** (*figura 2.7*).

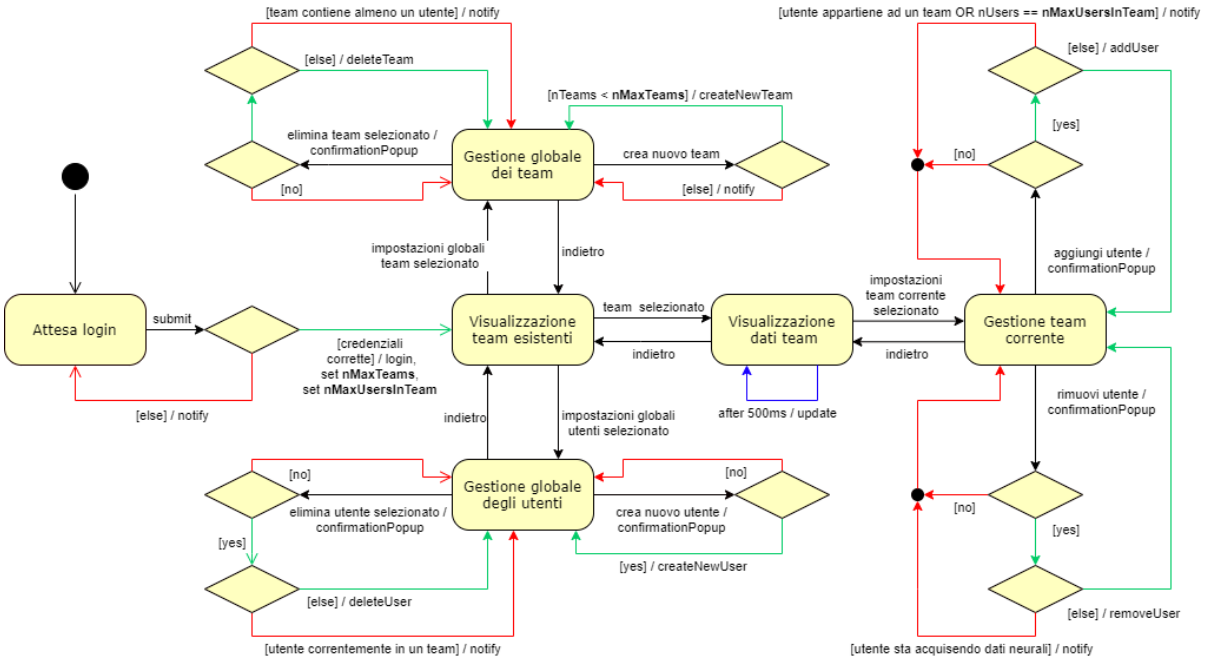

Figura 2.7: Analisi orientata agli stati - lato Dashboard

Una volta effettuato il login, il Team manager visualizza i team esistenti all'interno dell'organizzazione.
Da qui è possibile intraprendere tre strade. Due riguardano puramente aspetti gestionali:

- *Gestione globale dei Team*
  Qui il Team manager può creare nuovi Team (senza superare il limite imposto dalla licenza), oppure eliminare dei Team già esistenti.
  Prima di eliminare un Team è necessario rimuovere tutti i suoi componenti al fine di evitare inconsistenze nel comportamento dell'App di acquisizione.

- *Gestione globale degli Utenti*
  Qui invece è possibile creare nuovi utenti (il numero di Utenti esistenti non è limitato, a differenza del numero di componenti di un Team), o eliminare Utenti già esistenti.
  Per eliminare un Utente è prima necessario rimuoverlo da un eventuale Team a cui partecipa, per le stesse motivazioni espresse nel punto precedente.

La terza e ultima strada rappresenta il core dell'applicativo Dashboard: la visualizzazione dei dati neurali del Team.
Qui, a cadenza periodica di 500 ms, il sistema interroga il Database richiedendo i nuovi dati disponibili relativi ai componenti del Team selezionato.

Se prima vi era una gestione globale da questo stato è possibile transitare ad una gestione inerente solo al Team selezionato.
Sarà possibile aggiungere Utenti (che già non appartengano ad un altro Team) ai componenti del Team selezionato (sempre non superando i limiti imposti dalla licenza acquistata) oppure rimuoverli (solo nel caso non stiano correntemente trasmettendo dati).

2.3 Wireframe del sistema

Ora che tutti i sottosistemi sono stati modellati, è il momento di delineare dei wireframe al fine di ottenere delle linee guida grafiche riguardo quanto specificato fin ora.

Verranno presentati i wireframe relativi ai due endpoint offerti ai clienti: l'Applicativo di registrazione e l'Applicativo Dashboard.
La piattaforma Atlas di contro non necessita di una GUI in quanto server.

Per ogni applicativo considerato analizzeremo più wireframe, relativi agli aspetti affrontati nelle sezioni precedenti.

2.3.1 Applicativo di registrazione

Il primo wireframe (*figura 2.8*) sintetizza le componenti della **schermata di login**.

Troviamo le *credenziali* che permettono all'Utente di identificarsi in modo univoco all'interno del sistema: username (identificativo), password e nome dell'organizzazione.

In fondo è presente il pulsante che permette di effettuare il *login* una volta inserite le credenziali sopra citate.

Viene inoltre fornita la possibilità di *memorizzare le credenziali* dell'utente.

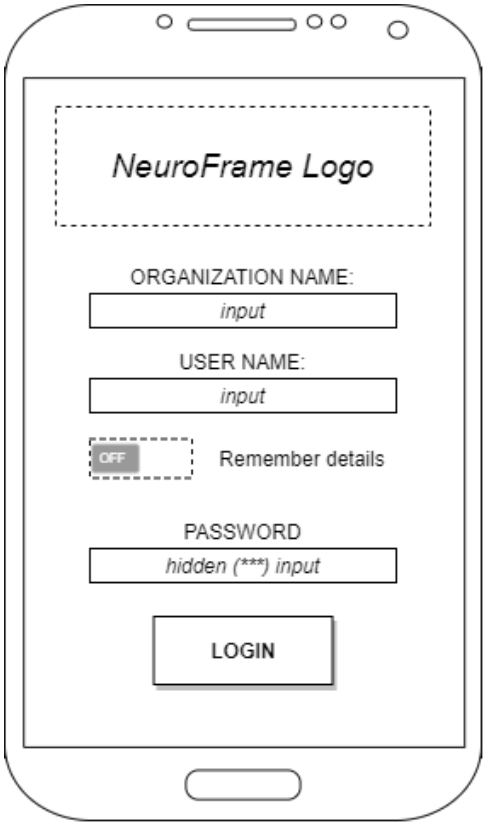

Figura 2.8: Pagina di login dell'Applicativo di registrazione

Il secondo wireframe invece (*figura 2.9*) contiene **tutte le rimanenti schermate**, sintetizzando un flusso funzionale derivato dall'analisi degli stati (*sezione 2.2.5*).

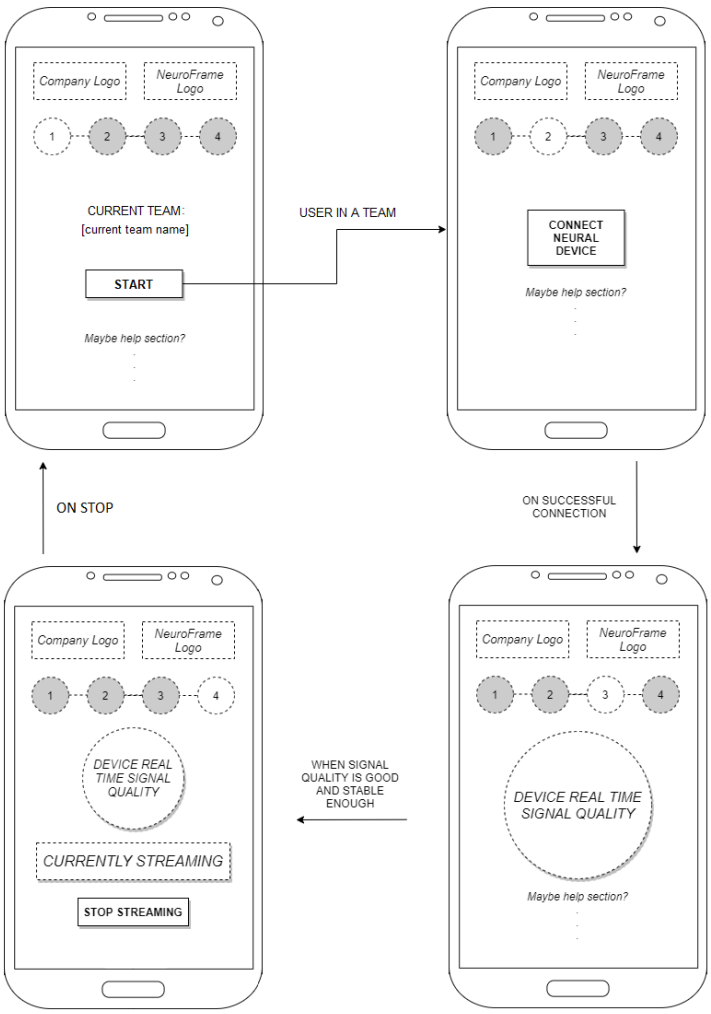

Figura 2.9: Flusso funzionale dell'applicativo di registrazione

Nella **schermata 1** l'applicazione attende l'input da parte dell'Utente per cominciare la procedura che porterà allo streaming di dati.
Come da modellazione non è possibile procedere alle prossime fasi fino a che l'Utente non è stato inserito all'interno di un Team da parte di un Team manager (il team corrente viene indicato nella schermata).

Giunti alla **schermata 2** l'applicativo si trova nuovamente in uno stato di attesa dell'input di un Utente.
Alla pressione del pulsante inizierà la procedura di connessione con il Dispositivo di acquisizione.

Una volta che la connessione è avvenuta con successo si viene portati alla **schermata 3** dove avviene la stabilizzazione del segnale al termine del quale comparirà la **schermata 4**, mostrando lo streaming in corso e fornendo all'Utente la possibilità di interrompere lo streaming, tornando alla schermata 1.

2.3.2 Dashboard

Sono stati elaborati due wireframe lato applicativo Dashboard: **Login** (*figura 2.10*) e **Dashboard visualizzazione dati** (*figura 2.11*).
Il primo wireframe presenta gli elementi standard di un qualsiasi form di login: un identificativo ed una password.
Si può notare l'*assenza* di due funzionalità di uso comune:

- *Registrazione*: data la natura di NeuroFrame come servizio in abbonamento alle aziende le credenziali verranno comunicate direttamente all'azienda, rendendo questa funzionalità superflua.
- *Credenziali dimenticate*: l'assenza di questa possibilità in una prima versione del sistema è da imputarsi alla prioritizzazione delle funzionalità essenziali al corretto funzionamento del sistema, anche se non si esclude l'implementazione di tale funzionalità in versioni future.

Figura 2.10: Pagina di login dell'applicativo Dashboard

Nel secondo wireframe possiamo individuare *tre macro aree*:

- **Impostazioni**
  In alto troviamo i componenti che permetteranno al Team manager di modificare varie impostazioni della visualizzazione a schermo e di gestire i componenti del team.

- **Area singoli componenti**
  Qui individuiamo i dati del benessere cognitivo relativi ad ogni componente del team.
  I suddetti dati verranno mostrati al Team manager attraverso modalità di rappresentazione coerenti con il tipo di dato, ad esempio con grafici e labels.

- **Area del team**
  Quest'area risulta analoga alla visualizzazione dei dati di un singolo utente ma descrive il benessere del Team nel complesso.
  Anche qui verranno utilizzate le stesse modalità di rappresentazione dei dati con aggiunta di label che descrivano dati di utilità (ad esempio il componente del Team più affaticato).

Analogamente alla schermata di login si vuole mostrare a schermo il logo dell'organizzazione fruitrice del servizio, affiancato al logo di Vibre.

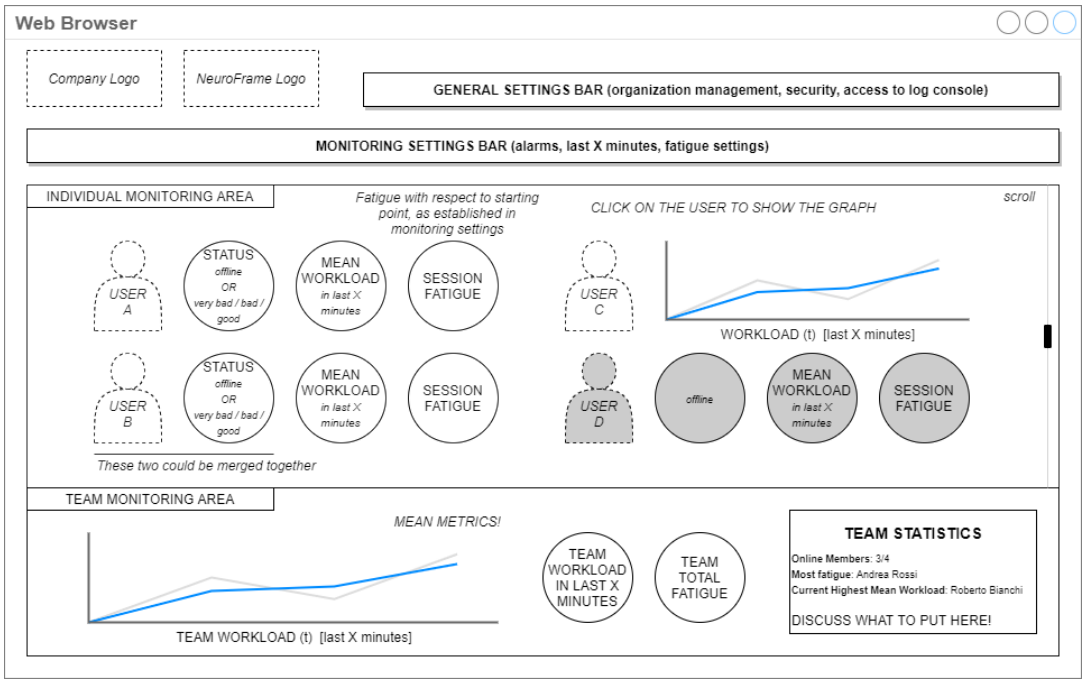

Figura 2.11: Schermata Dashboard visualizzazione dati

2.4 Aspetti critici dell'applicativo Dashboard

Lo natura di questo applicativo pone davanti a delle problematiche e a conseguenti decisioni durante lo sviluppo.
Alcune problematiche risultano *legate intrinsecamente alla tecnologia implementativa*, cioè lo sviluppo di una soluzione che fa uso di tecnologie web (come specificato nei requisiti Should have *sezione 2.1.3*) piuttosto che un applicativo nativo.
Altre problematiche d'altro canto derivano da scelte *legate alla user experience*.
Segue una descrizione dei principali aspetti critici affrontati.

2.4.1 Risorse computazionali limitate dei browser

Scegliendo un approccio web piuttosto che un'applicazione nativa otteniamo un impoverimento dell'efficienza di calcolo, aggiungendo un layer extra tra l'applicativo e le risorse della macchina che lo eseguirà: il browser.
Le prestazioni dell'applicazione dunque non saranno influenzate solamente dalla capacità di calcolo della macchina, ma anche dall'ottimizzazione del browser, le risorse che il sistema operativo concederà ad esso e da fattori non predicibili (come limitazioni imposte dal browser stesso o un alto numero di schede aperte nel browser).
Al costo di una maggiore scalabilità e una compatibilità con un vasto parco dispositivi dunque si ottiene una riduzione delle performance che vincola ad uno sviluppo il più possibile ottimizzato.

2.4.2 Gestione di grandi quantità di dati

La quantità di dati che descrivono l'evolversi dello stato cognitivo di una persona nel tempo presentano un volume e una densità molto elevati, in quanto l'hardware che legge gli impulsi neurali (il caschetto Muse) effettua campionamenti del segnale con un periodo molto basso.
Questa problema diventa ancora più preponderante all'aumentare del numero di componenti di un team.
Risulta quindi necessario agire su più fronti:

- *Ottimizzare al massimo gli algoritmi di processing dei dati nella dashboard*, così da processare velocemente i dati prima di richiedere al database un nuovo aggiornamento
- *Ottimizzare le query di richiesta dati lanciate dalla dashboard verso il database*, al fine di ridurre il più possibile il tempo impiegato dal database ad eseguire la ricerca e per minimizzare la quantità di dati inviati nella rete
- *Limitare il numero massimo di componenti di un team*

- *Limitare se necessario la frequenza con cui l'app di acquisizione invia i dati alla piattaforma per l'elaborazione* scegliendo un buon compromesso tra accuratezza dei dati e performance

2.4.3 Rendering dei grafici

Questa problematica risulta strettamente correlata alla precedente.
Mentre in precedenza il problema verteva maggiormente sui calcoli effettuati in locale sui dati ricevuti, qui si vuole sottolineare come *il rendering dei grafici risulti molto oneroso computazionalmente.*
Risulta quindi fondamentale un'*ottimizzazione nella struttura e negli algoritmi che coinvolgono i dati utili al rendering.*

2.4.4 Design vertically constrained

La user interface che si occupa di mostrare i dati all'utente (la schermata dashboard) risulta vincolata da un requisito importante per l'efficacia pratica del sistema e per l'esperienza utente: *al fine di permettere al team manager di poter interpretare velocemente la condizione del team è necessario che la maggior parte dei dati utili sia sempre visibile a schermo.*
Questo ci porta ad avere delle costrizioni verticali (in altri termini tutto il contenuto verticale della pagina deve essere contenuto a schermo).
Le tecnologie in ambito web utilizzano come punto di riferimento l'asse orizzontale della pagina, forti del fatto di poter estendere la pagina verticalmente potenzialmente all'infinito per poter rappresentare il contenuto.
Risultano dunque necessarie delle soluzioni ad hoc per soddisfare il requisito preposto.
In particolare, si è optato per tenere sempre visibili a schermo la barra delle opzioni, l'area dei singoli utenti e l'area del team nel complesso. Essendo il numero di componenti del team imprevedibile a priori, si è scelto di far scrollare verticalmente il contenuto dell'area degli utenti.

Capitolo 3

Progettazione del sistema

In questo capitolo affronteremo aspetti inerenti alla progettazione di NeuroFrame.
In primo luogo vedremo l'approccio utilizzato per lo sviluppo di questo progetto: *il modello Agile*; inoltre chiariremo quali tra le specifiche del sistema sono state considerate per la prima versione del progetto.
Dopo di che verrà discussa la tipologia di sistema di NeuroFrame, cioè un *sistema distribuito*. In particolare saranno approfonditi concetti quali Microservizi e Software as a service.
Scenderemo poi di livello rispetto ai capitoli precedenti esaminando l'effettiva *architettura del sistema NeuroFrame*. In questa sezione saranno anche affrontati *la tecnologia utilizzata per il Database e la struttura interna di quest'ultimo.*

3.1 Sviluppo attraverso un modello Agile

Quanto affrontato fin ora segue una metodologia di sviluppo chiamata **modello Agile**. Questa metodologia di lavoro poggia le sue fondamenta nei principi dell'antecedente scuola dell'**eXtreme programming**.

La filosofia proposta dal modello Agile è, a mio avviso, ben riassunta in questo estratto:

> *"The keys to modeling success are to have effective communication between all project stakeholders, to strive to develop the simplest solution possible that meets all of your needs, to obtain feedback regarding your efforts often and early, to have the courage to make and stick to your decisions, and to have the humility to admit that you may not know everything, that others have value to add to your project efforts" [11].*

Traducendo questi principi in azioni concrete risulta fondamentale **un'attenta modellazione e documentazione del lavoro**; il processo di modellazione e sviluppo va intrapreso tenendo a mente un **approccio incrementale** che si sviluppa in iterazioni più o meno numerose.

In modo particolare risulta necessario produrre un'ingente quantità di **artefatti** (schemi, diagrammi, ecc...) per cogliere anzitempo gli aspetti più complessi ed evitare di incappare in gravi errori una volta giunti alle fasi successive.
Gran parte dei rallentamenti nei lavori e conseguenti costi aggiuntivi sono dovuti ad una progettazione molto sommaria.

Di contro però, proprio per via della natura iterativa di questa metodologia, è fondamentale progettare un sistema preposto a futuri cambiamenti e di facile espandibilità; bisogna dunque evitare il fenomeno opposto alla sommarietà: l'**overfitting**.
Durante il processo è necessario ottenere quanto più **feedback** possibile da utenti target che utilizzeranno il servizio.

Nonostante questa sia ancora la prima iterazione del progetto, Vibre ha già effettuato lavori di confronto con la clientela interessata a NeuroFrame al fine di ottenere feedback utili, in particolare è stata presentata al cliente una **Proof Of Concept** (POC) sulla cui base sono stati dati feedback e proposte potenziali features, fra queste per esempio è emersa la necessità di un sistema di allarme in caso un soggetto dia segni di comportamenti anomali (affaticamento prolungato, sonnolenza, stress per lunghi periodi di tempo...).

Sono stati anche svolti degli incontri con un pilota professionista di auto sportive immerso nel contesto di un simulatore di guida; l'obiettivo è stato, oltre che la costruzione del prodotto, la validazione delle metriche contestualizzate a un caso d'uso specifico e reale.

Un'altra azione pratica da intraprendere, di particolare importanza durante le prime iterazioni, è **mantenere il sistema semplice**, andando a ricercare solo le feature core che si vogliono offrire agli utenti; per questa ragione non tutte specifiche elencate nella *sezione 2.1* risulteranno implementate al momento, in modo particolare in riferimento all'applicativo Dashboard, focus della mia esperienza di tirocinio in azienda.

3.1.1 Feature ricercate nella versione base del sistema durante la mia esperienza di tirocinio

Dati i principi discussi nella sezione precedente, la natura del progetto e la mole di lavoro che esula enormemente dall'ammontare ore di questa esperienza di tirocinio, **abbiamo scelto di concentrare i nostri sforzi nei requisiti di sistema definiti come Must have** (*sezione 2.1.2*).

In modo particolare i miei sforzi lavorativi sono stati volti verso la **realizzazione dell'applicativo Dashboard**, tenendo a mente le prospettive di futura evoluzione del progetto creando un sistema il quanto più possibile preposto a soddisfare facilmente i requisiti di sistema secondari non implementati in questa versione base.

Come vedremo più avanti, **tutti i requisiti must have relativi alla Dashboard hanno trovato la loro realizzazione**, ottenendo un prodotto le cui funzionalità core sono già operative.

Data la momentanea assenza di un applicativo di acquisizione sviluppato e il continuo raffinamento degli algoritmi neurali all'interno della piattaforma Atlas, per completare e validare il mio lavoro ho inoltre implementato un **simulatore esterno** al fine di poter verificare il funzionamento di quanto da me sviluppato; l'argomento simulatore verrà affrontato nel *capitolo 5*.

3.2 NeuroFrame come sistema distribuito

Andrew S. Tanenbaum e Marteen Van Steen nel loro libro *"Sistemi distribuiti"*, definiscono ciò che dà titolo al loro trattato come segue:

> *"Un **sistema distribuito** è una collezione di computer indipendenti che appare ai propri utenti come un singolo sistema coerente [12]."*

Dalla *sezione 2.2.2*, dove vengono descritti i sottosistemi coinvolti, è già possibile intuire che la natura di NeuroFrame è prettamente quella di un sistema distribuito; ogni sottosistema descritto difatti opera su di un calcolatore indipendente in un rapporto di collaborazione con le altre entità perpetrato scambiando messaggi su interfacce di comunicazione definite.

Dal punto di vista degli utenti finali l'Applicativo di registrazione e l'Applicativo Dashboard operano coerentemente in un rapporto causa-effetto, partendo dalla lettura di un segnale neurale fino alla visualizzazione di tale segnale elaborato come metrica di stato

cognitivo nella Dashboard di controllo.

Impostato NeuroFrame come sistema distribuito, è possibile introdurre due concetti collegati e che definiscono la natura del servizio:

- La filosofia alla base della piattaforma Atlas, i **microservizi**.
- La modalità con la quale il cliente finale può fruire del servizio, la metodologia **software as a service**.

3.2.1 Microservizi

Come introdotto nella *sezione 1.3*, la piattaforma Atlas (*figura 1.4*) è composta dal NeuroServer (server con il compito di gestire l'invio dei messaggi in entrata ed in uscita) e dagli Algorithm Packages.

Rivolgendo l'attenzione ai pacchetti degli algoritmi troviamo:

- *MindFeel*
- *MindPulse*
- *MindPrint*

In particolare, all'interno del sistema NeuroFrame, l'elaborazione delle metriche è resa possibile solamente grazie allo sviluppo dell'algoritmo MindPulse; analogamente, in altri progetti aziendali, i restanti algoritmi sono alla base del servizio offerto.

Questa suddivisione in unità il più possibili atomiche e semplici corrisponde ad un **approccio a microservizi** [13].

L'architettura a microservizi punta a creare una collezione di servizi che siano quanto più possibile:

- *manutenibili e testabili*
- *debolmente dipendenti l'uno con l'altro*
- *distribuibili indipendentemente*
- *organizzati seguendo il valore di business che il servizio vuole offire*
- *gestiti da piccoli team*

I microservizi sviluppati poi possono essere facilmente integrati in progetti più grandi che ne sfruttano le feature implementate.

Il microservizio dell'algoritmo MindPulse, reso accessibile attraverso NeuroServer, ben si sposa all'interno del sitema distribuito NeuroFrame ed offre il vantaggio di essere riutilizzabile in qualsiasi progetto futuro essendo difatti l'elaborazione delle metriche totalmente svincolata da NeuroFrame.

L'indipendenza degli algoritmi dai servizi che ne richiedono l'elaborazione dati è data dalla caratteristica per cui l'unico requisito necessario per il funzionamento di un algoritmo è fornire un input da elaborare e trasformare in output, indipendentemente da chi fornisca tali dati.

3.2.2 Software as a service

L'essere un sistema distribuito e la volontà (espressa nella *sezione 2.1.3*) di rendere gli applicativi utente fruibili tramite applicazioni web, rende NeuroFrame adatto ad una commercializzazione come **software as a service**.

> *"Software-as-a-Service (SaaS) è un modello di licenza e distribuzione utilizzato per fornire applicazioni software su Internet - dunque come servizio [14]."*

Questa volontà era stata già sottointesa nell'analisi orientata agli oggetti (*sezione 2.2.4*), dove per ogni Organizzazione è necessario memorizzare la licenza acquistata (con degli effetti sul servizio dipendenti dalla scelta del tipo di licenza).

Già nei primi anni '60 l'idea di centralizzare l'hoisting di servizi aveva preso forma in quello che era al tempo denominato **Application server provider** (*ASP*).

Con l'evolversi delle infrastrutture tecnologiche della rete ed una conseguente diffusione di essa, anche in contesti diversi da quelli aziendali, il software as a service è diventato un modo estremamente diffuso di fruire di servizi informatici.

Si pensi semplicemente, in ambiente consumer, a servizi quali Netflix, Spotify, Amazon Prime, ecc...

Ciò che ha reso vincente questo approccio è l'alta scalabilità, derivante dall'estrema semplicità di fruizione dei servizi; infatti, essendo diffusi tramite web, tali servizi non nececessitano di un'installazione locale. Grazie a ciò risultano facili e veloci da utilizzare per qualsiasi target di utenza.

3.3 Architettura del sistema NeuroFrame

Analizziamo ora l'architettura dell'intero sistema NeuroFrame scendendo di livello. La *figura 3.1* in questa sezione espande quanto visto ad alto livello nella *figura 2.2*.

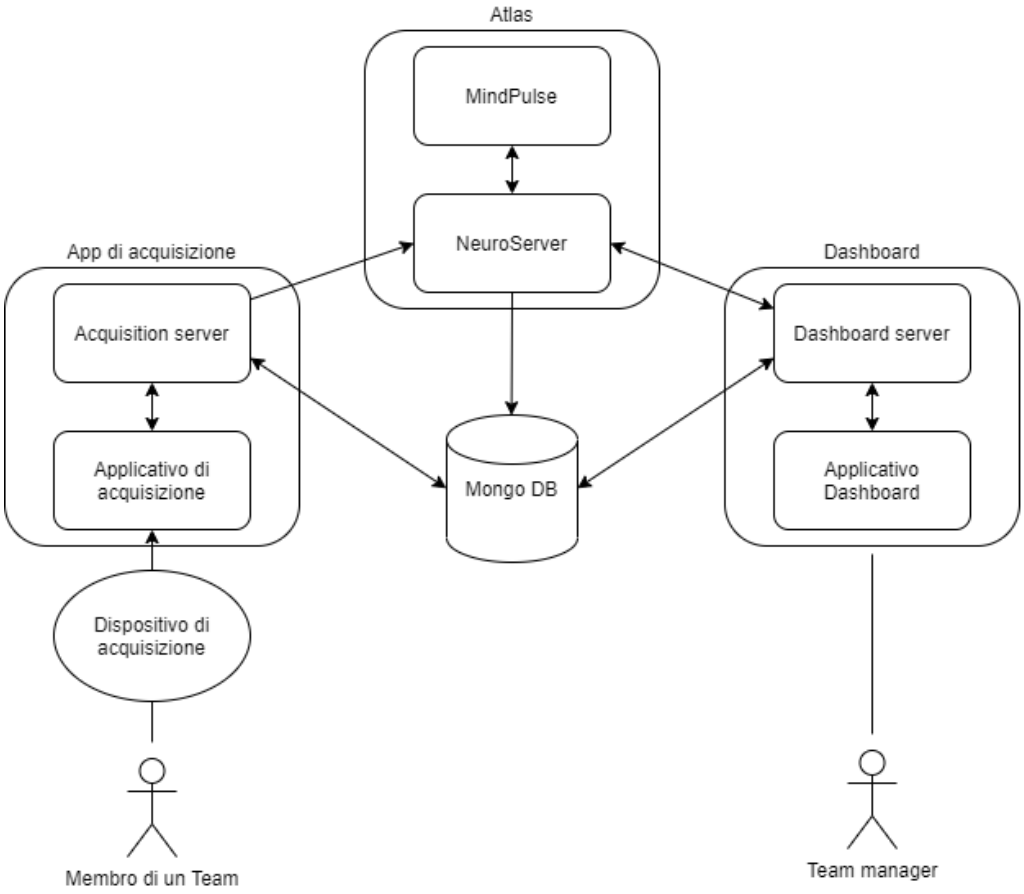

Figura 3.1: Architettura completa del sistema distribuito NeuroFrame

Qui ritroviamo la nota struttura interna della **piattaforma Atlas**, costituita dal **NeuroServer** e dagli *algoritmi di elaborazione* (in figura per semplicità è indicato l'unico algoritmo rilevante nel sistema NeuroFrame, cioè **MindPulse**).

Anche lato Acquisizione e Dashboard troviamo una divisione analoga; **per ogni endpoint vi è un applicativo usato dagli utenti finali ed un server dedicato**. I server comunicano tra loro ed utilizzano il database come snodo centrale per condividere dati persistenti.

Riprendendo il flusso informativo lato Dashboard, discusso nella *sezione 2.2.3*, possiamo vedere in *figura 3.2* come esso rimanga sostanzialmente invariato se non per il fatto che vi è un passaggio extra dato dal server (la situazione è analoga lato Acquisizione, che il cui diagramma di sequenza non viene riportato per evitare eccessiva ridondanza).

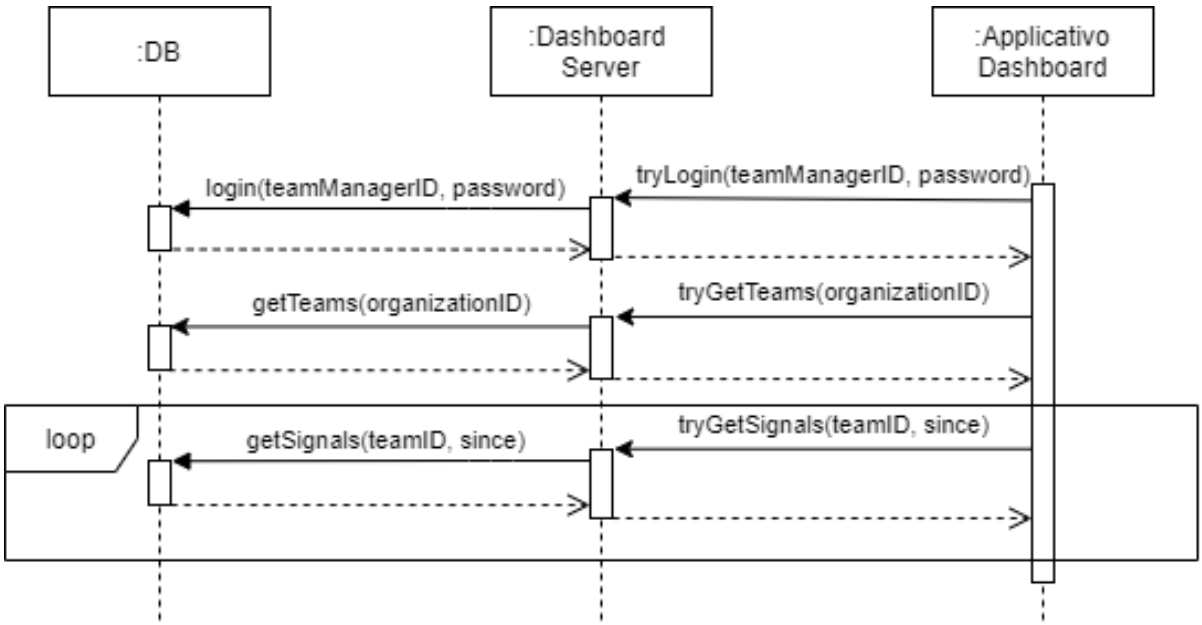

Figura 3.2: Diagramma di sequenza rappresentante il flusso informativo completo lato Dashboard

Vediamo ora le **scelte tecnologiche e strutturali legate al database**.

3.3.1 Storage dei dati: utilizzo di database non relazionali

Il database scelto per NeuroFrame è di tipo **non relazionale**, in particolare si è optato per l'utilizzo di **MongoDB** [15].
La scelta è stata dettata dalle caratteristiche del mezzo che ben si sposano con le esigenze di Vibre e del progetto.

In primo luogo la motivazione più importante che ha portato a tale scelta è da attribuirsi alla grande **flessibilità** offerta da MongoDB; questa caratteristica ricopre un ruolo fondamentale all'interno di un progetto in rapido mutamento.
Se difatti un classico database relazionale SQL offre dei vincoli che rendono la consistenza dei dati molto solida, le stesse limitazioni mal si sposano con l'evolversi di nuove

feature (dettate anche dal feedback dei clienti) e delle strutture dati stesse (specialmente in vista delle evoluzioni degli algoritmi neurali della piattaforma Atlas, in particolare dell'algoritmo MindPulse in questo contesto).

MongoDB inoltre utilizza documenti **JSON-like**, un formato semplice da usare che negli ultimi anni si sta rapidamente affermando come standard de facto, specialmente in ambito web.

Ultimo motivo, non meno importante, è la capacità di MongoDB di **eseguire interrogazioni e manipolazioni dei dati lato DB molto più potenti e complesse** rispetto ad un database relazionale SQL.

3.3.1.1 Struttura del database

Di seguito sono esposte le *collezioni* che compongono il database MongoDB dedicato al sistema NeuroFrame, nonché la *struttura interna* di ciascuna di esse.

Organizations
Collezione dedicata allo storage delle organizzazioni che utilizzano il servizio.

- *organization id*
  Identificativo, univoco per ogni organizzazione.
- *logo*
  URI preposto al recupero del logo dell'organizzazione.
- *teams*
  Array contenente gli identificativi dei Team presenti nell'organizzazione.
- *team managers*
  Array contenente gli identificativi dei Team managers presenti nell'organizzazione.
- *users*
  Array contenente gli identificativi degli Utenti presenti nell'organizzazione.
- *license*
  Struttura volta ad indicare i dati relativi al tipo di licenza acquistata (si vedano vincoli definiti nella sezione *2.2.4*).
  - *name*
    Tipologia di licenza acquistata.
  - *max users per team*
    Numero massimo di Utenti *per ogni Team* consentito dal tipo li licenza acquistata.

  - *max teams*
    Numero massimo di Team *nell'Organizzazione* consentito dal tipo di licenza acquistata.

Team managers
Qui vengono memorizzati tutti i Team manager esistenti.

- *user id*
  Identificativo, univoco *all'interno della stessa Organizzazione*.
- *organization id*
  Riferimento all'identificativo dell'Organizzazione di cui il Team manager fa parte.
- *password*
  Password del Team manager.

Users
Collezione preposta alla memorizzazione degli Utenti presenti in NeuroFrame.

- *user id*
  Identificativo dell'Utente, *univoco all'interno della stessa Organizzazione*.
- *organization id*
  Riferimento all'Organizzazione di cui l'Utente fa parte.
- *password*
  Password dell'Utente.
- *current team*
  Riferimento all'identificativo del Team di cui l'Utente fa parte al momento (vuoto se non fa parte di un Team).

Teams
Spazio dove saranno memorizzati i dati relativi a tutti i Team.

- *team id*
  Identificativo del Team, *univoco all'interno dell'Organizzazione*.
- *organization id*
  Riferimento all'identificativo dell'organizzazione che ha creato il Team.
- *users*
  Array contentente gli identificativi degli Utenti facenti parte del Team.

Signals
Collezione chiave per il funzionamento di NeuroFrame. Contiene tutte le metriche elaborate dalla piattaforma Atlas.
Ogni entry all'interno della collezione contiene un numero massimo di misurazioni, date delle limitazioni imposte sulle dimensioni dei dati da MongoDB. Una stessa acquisizione di segnali neurali dunque, dipendentemente dalla sua durata temporale, può essere divisa in più parti; ogni parte corrisponde a singole entry di questa collezione.

- *user id*
  Riferimento all'identificativo dell'Utente a cui le metriche fanno capo.
- *organization id*
  Riferimento all'identificativo dell'Organizzazione a cui l'Utente appartiene.
- *team id*
  Riferimento al Team di cui l'Utente era membro *nel momento dell'acquisizione* (è possibile che il Team manager sposti l'Utente in un altro Team).
- *data*
  Array contenente dati strutturati ordinati temporalmente.
  - *fatigue*
    Metrica rappresentante l'affaticamento mentale dell'individuo (*sezione 1.1*).
  - *workload*
    Metrica rappresentante il carico cognitivo dell'individuo (*sezione 1.1*).
  - *timestamp*
    Istante temporale del campionamento.
- *start*
  Corrisponde al timestamp del primo elemento dell'array data.
- *end*
  Corrisponde al timestamp dell'ultimo elemento dell'array data.

Capitolo 4

Sviluppo dell'applicativo Dashboad real-time

Analizziamo la fase di sviluppo del sottosistema Dashboard all'interno del macrosistema Neuroframe.
Verranno innanzitutto esplicate le tecnologie coinvolte ed i loro principi di funzionamento.
Infine verrà esposta l'architettura vera e propria del progetto.

4.1 Tecnologie coinvolte

Essendo questo un applicativo web-based, sono state scelte delle tecnologie e API particolari in grado di svolgere questo compito.
In particolare data la natura medio-ampia del progetto, che esula dai compiti più "tradizionali" di una pagina web, risultano necessarie soluzioni che permettono di ottenere una struttura solida e ben organizzata; quest'ultima si deve adattare ad una modellazione orientata agli oggetti, per meglio dividere le responsabilità delle componenti e, soprattutto, mantenere ed aggiornare lo stato di ogni componente ed i dati che da esso verranno utilizzati.
Per soddisfare tali necessità si è deciso di optare per due framework ben consolidati: **React** e **Redux**.
Attraverso queste due soluzioni risulta possibile trarre il massimo da JavaScript (in particolare *Node Js*) con soluzioni semplici ed efficaci.

4.1.1 React

React, come citato sopra, permette di *modellare il nostro codice seguendo i principi della programmazione ad oggetti.* In modo particolare risulta essenziale comprendere quattro concetti chiave:

- **JSX**
  JSX [16] è un'estensione della sintassi di JavaScript.
  Permette di unire gli aspetti di html come linguaggio di template agli aspetti di JavaScript come linguaggio di scripting in una forma che ne aumenta semplicità e leggibilità.
  Gli elementi JSX vengono utilizzati nelle definizioni delle funzioni di rendering (che verranno trattate a breve) semplificando la costruzione della user interface.
  Attraverso JSX possiamo richiamare un componente React tramite con un meccanismo analogo ai tag in html.

- **Componenti**
  Attraverso un componente [17] andiamo a definire quella che risulta essere a tutti gli effetti una classe.
  Il nostro componente, definito da uno o più costruttori, contiene uno stato; questo stato verrà mantenuto, ed eventualmente aggiornato, durante tutto il ciclo di vita (lifecycle) del componente.
  È possibile ricevere dati ed istruzioni da altri componenti attraverso le props.

- **Stato**
  Lo stato [18] un insieme di proprietà di un componente.
  Queste proprietà possono variare a seguito dell'interazione con altri componenti o come azione del componente stesso nel caso esso esegua delle azioni a cadenza temporale.
  In React inoltre lo stato risulta fondamentale ai fini di ottenere un rendering a schermo performante; ogni componente React infatti deve obbligatoriamente definire una funzione *render()*.
  Attraverso di essa verrà ritornato il contenuto da renderizzare a schermo.
  React cambierà il contenuto a schermo (consumando quindi risorse) solamente quando vi saranno delle modifiche nel contenuto ritornato dalla funzione render(); utilizzando dunque le proprietà che definiscono lo stato del componente all'interno della funzione render() del componente stesso riusciremo a ridurre al minimo indispensabile il numero di volte in cui l'applicazione verrà renderizzata, riflettendo i cambiamenti di stato del componente.

- **Lifecycle**
  All'interno del nostro componente possiamo (opzionalmente) definire due funzioni:

  - *componentDidMount()*
    Questa funzione sarà richiamata quando il componente verrà istanziato.

È qui che ad esempio verranno definiti dei timer attraverso i quali richiamare periodicamente delle funzioni.

- *componentDidUnmount()*
  Analogamente questa funzione essa verrà richiamata quando il componente verrà distrutto.
  Sarà dunque qui che ad esempio i timer periodici verranno fermati.

4.1.2 Redux

Redux [19] è un *contenitore di stato* per applicazioni JavaScript.
Gode di quattro caratteristiche fondamentali per progetti portata medio-grande:

- **Deterministico**
  Un aspetto fondamentale nelle applicazioni web di ogni genere (e soprattutto in un contesto real-time) è il determinismo.
  L'oneroso compito di far collaborare tutti i componenti al fine di ottenere un comportamento predicibile viene largamente semplificato dal framework Redux.

- **Centralizzato**
  Avere stato e logica centralizzati permette di ottenere rapidamente delle feature di fondamentale importanza, come funzioni di "annulla" e "ripeti" che ci permettono di muoverci agilmente tra lo storico degli stati.
  Inoltre un approccio centralizzato garantisce un consistenza maggiore dei dati, rendendo possibile modificarli solamente attraverso specifiche funzioni (azioni) create durante la definizione della struttura dati.

- **Debug oriented**
  Attraverso semplici plugin browser come Redux DevTools risulta immediato il debug dell'applicazione.
  Difatti attraverso questi tool otteniamo una visione completa dello stato dei dati e della sua evoluzione nel tempo, riuscendo ad identificare con precisione quando e soprattutto perchè lo stato abbia subito delle modifiche.

- **Flessibile**
  Redux funziona con ogni layer di User Interface e, essendo ormai uno strumento consolidato, dispone di un solido supporto e un vasto parco di plugin e pacchetti aggiuntivi per ogni esigenza.

Per comprendere come queste caratteristiche si concretizzino risulta fondamentale comprendere tre concetti chiave nella struttura di Redux:

- **Store**
  Lo store [20] è un oggetto che contiene l'intera struttura ad albero dello stato.
  Fornisce metodi per la lettura dello stato corrente.

- **Reducer**
  I reducer [21] definiscono la struttura dello store.
  Vi possono essere più reducer all'interno di una stessa applicazione, al fine di meglio suddividere lo stato.

- **Azioni**
  Le azioni [22] permettono di modificare il contenuto dello store.
  Essendo queste l'unico modo per alterare lo stato attuale, qualsiasi componente che voglia agire sullo stato deve passare per le azioni definite.

Ma *perchè* utilizzare un contenitore di stato se ogni componente React possiede già uno stato? React, pur essendo uno strumento molto potente, ci pone davanti a delle limitazioni:

- Innanzitutto non è raro che più componenti debbano fare riferimento allo stesso dato; la presenza di uno stato comune evita di dover implementare meccanismi ad hoc di sincronia tra gli stati dei due componenti.
  Lo stato di ogni componente, attraverso tool ready to use, sarà dunque sincronizzato con la struttura principale.

- In secondo luogo React incoraggia lo scambio di dati solo in una direzione: da un componente padre verso un componente figlio (utilizzato dunque dal padre).
  La presenza delle azioni Redux risolvono questo problema in quanto ogni cambiamento apportato allo stato Redux si rifletterà su tutti i componenti React che utilizzano quel particolare dato.

Redux però ci pone davanti delle complicanze di carattere pratico.
Nella sua versione base questo framework ci pone davanti ad una *ingente quantità di boilerplate code*, difatti sono necessarie ingenti righe di codice con pochissime differenze per definire reducer e azioni.
Inoltre risulta particolarmente ostica la creazione di middleware per la gestione di cambiamenti asincroni nello stato.

Per ovviare a questo problema si è scelto di integrare l'utilizzo di *Redux Toolkit* [23], che automatizza questo processo permettendoci di definire una struttura di più alto livello (*Slice*) che viene poi convertita nelle strutture sopra citate.
In particolare i middleware per i cambiamenti di stato asincroni, attraverso Redux Toolkit, vengono identificati dai *Thunk* [24].
Utilizzando questa soluzione si è ottenuto un approccio per le azioni asincrone sicuro e si è evitato di incrementare ingentemente la quantità di codice del progetto, mantenendo una struttura ordinata e facilmente manutenibile.

4.2 Architettura del sistema Dashboard real-time

In questa sezione discuteremo della struttura implementativa dell'applicativo e delle soluzioni utilizzate.
Il progetto viene suddiviso in due sotto-applicativi:

- **Applicativo Dashboard**
- **Dashboard server**

La ragione di tale scelta è da individuarsi nelle richieste dati asincrone da parte dell'applicativo Dashboard: come discusso in precedenza il meccanismo che ci viene fornito da Redux per eseguire aggiornamenti asincroni è chiamato Thunk.
Ogni Thunk asincrono all'interno dell'applicativo Dashboard invierà una richiesta POST tramite protocollo HTTP al Dashboard server (che difatto implementa un server Node.js) il quale si occuperà di interrogare il database MongoDB secondo i parametri specificati dall'applicativo Dashboard e ritornerà il risultato (*figura 4.1*).

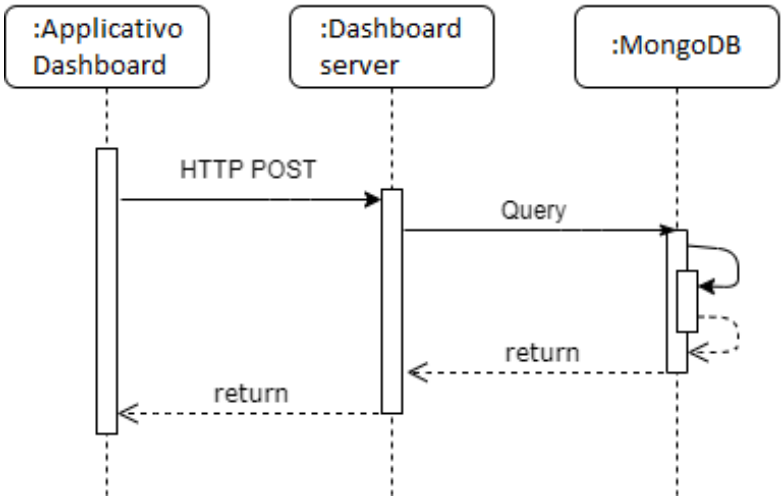

Figura 4.1: Interazione tra l'applicativo Dashboard, Dashboard server e MongoDB

Affrontiamo ora nel dettaglio struttura e funzionamento dell'applicativo Dashboard e di Dashboard server.

4.2.1 Applicativo Dashboard

Come struttura del progetto è stato utilizzato un approccio *feature based*; tale approccio organizza i file in due categorie:

- **Feature**
  Una feature identifica una funzionalità offerta dal sistema.
- **Componenti**
  Un componente viene utilizzato da una feature per eseguire la sua funzionalità o, alternativamente, da altri componenti.

Il progetto risulta quindi strutturato come segue:

```
src
├── components
│   ├── common.css
│   ├── CommonComponent1.js
│   ├── CommonComponent2.js
│   └── ...
└── features
    ├── feature1
    │   ├── Feature1Component1.js
    │   ├── Feature1Component2.js
    │   ├── ...
    │   ├── feature1Slice.js
    │   └── feature1.css
    ├── feature2
    │   └── ...
    ├── store
    │   ├── index.js
    │   └── reducers.js
    ├── utils
    ├── tests
    └── index.js
```

Possiamo notare che per ogni feature viene creato un file Slice.
Come anticipato precedentemente (*sezione 4.2.2*), attraverso lo Slice (fornitoci da Redux Toolkit) è possibile definire comodamente reducers, azioni sincrone e azioni asincrone (Thunk).
Nella cartella *store* inoltre troviamo il file *reducers.js*, attraverso il quale combineremo i reducers contenuti in ogni Slice al fine di creare lo Store (contenuto nel file index.js).

Al fine di una maggiore chiarezza si ricorda che feature e componenti vedono realizzata la loro implementazione attraverso React.
Dashboad app risulta composta da tre feature:

- **Login**
- **Home**
- **Dashboard**

Analizziamo ora nel dettaglio compiti e funzionalità di ogni feature.

4.2.1.1 Login

Corrisponde al servizio di login identificato nel primo wireframe della *sezione 2.3.2* (*figura 2.10*).
Trattandosi di una feature semplice non è risultato necessario implementare componenti a supporto di questa feature.
Al fine di ottenere una user interface gradevole per l'utente finale è stata utilizzata una libreria grafica di supporto a React chiamata *Ant design* [25]; questa libreria viene dunque utilizzata in gran parte delle feature e dei componenti.

4.2.1.2 Home

A login effettuato il Team manager si troverà nella home.
Qui vengono individuate due aree corrispondenti ad altrettanti componenti:

- *TopMenu*
  In cima alla pagina troviamo una barra contenente dei bottoni i quali, una volta premuti, permettono all'utente di accedere alle impostazioni generali di gestione (realizzate attraverso un modale a comparsa).
  Sarà dunque possibile creare, modificare ed eliminare utenti da monitorare ed inserire all'interno di Team.
  Analogamente le stesse operazioni di creazione, modifica ed eliminazione saranno possibili con i Team stessi.

- *Teams*
  In questa sezione invece verranno visualizzati i Team creati dai Team manager attraverso il TopMenu precedentemente discusso.
  Per ogni Team verrà visualizzato il nome del Team ed una lista di tutti i componenti.
  Il Team manager, interagendo con un Team, accederà alla feature Dashboard.

4.2.1.3 Dashboad

La feature Dashboard, oltre a rappresentare il punto cardine dell'applicativo, risulta essere anche la più complessa.
Sulla falsa riga del wireframe precedentemente discusso (*figura 2.11*) risultano individuati tre componenti principali:

- *Settings*
  Qui il Team manager potrà tornare alla Home, visualizzare il nome del team ed accedere alle impostazioni del team.
  Queste ultime sono state divise a loro volta in due componenti più semplici:

  - *UserSettings*
    Qui il Team manager potrà aggiungere e rimuovere utenti dal team.
    È importante sottolineare nuovamente che un Utente non potrà inviare segnali verso la piattaforma attraverso l'App di registrazione finchè un Team manager non l'avrà inserito in un team.
  - *DashboardSettings*
    Attraverso questo componente il Team manager sarà in grado di modificare delle impostazioni strettamente legate alla visualizzazione dei dati, ad esempio modificando l'intervallo temporale delle informazioni mostrate a schermo.

- *UserArea*
  In quest'area saranno visualizzati i dati relativi ad ogni componente del team.
  Vi sono tre informazioni mostrate a schermo, ognuna a sua volta implementata come un componente:

  - *ConnectionStatus*
    Descrive lo stato della connessione dell'Utente.
    Nella prima versione dell'applicativo lo stato della connessione risulta binario (online e offline), ma in versioni future verrà indicata anche la qualità del segnale inviato (corrispondente ad un posizionamento ottimo o pessimo del caschetto Muse).
  - *MeanWorkload*
    Attraverso un grafico a torta viene indicato il valore medio del carico cognitivo dell'Utente.
    Trattandosi di un valore compreso in un intervallo chiuso da -1 a 1 il grafico si riempirà in senso orario all'aumentare di valori positivi ed in senso antiorario al diminuire di valori negativi, differenziando anche il colore in base al segno. Inoltre, superata una certa soglia critica pericolosa per l'Utente il colore del grafico varia ulteriormente, segnalandolo all'attenzione del Team manager.

– *SessionFatigue*
L'affaticamento nella sessione di lavoro viene espresso attraverso lo stesso grafico a torta esplicato sopra ma in un range di valori differenti.

Interagendo con l'area di un Utente verrà cambiata modalità di visualizzazione, mostrando in un grafico cartesiano i singoli valori del carico cognitivo all'interno dell'intervallo temporale definito in DashboardSettings.

- *TeamArea*
Analogamente alla UserArea, nella TeamArea verranno mostrate le stesse informazioni sopra descritte ma rappresentanti la media di tutto il team.
Vengono riutilizzati i componenti dei grafici a torta e cartesiano, aggiungendo un nuovo componente *TeamStatistics* dove vengono indicate alcune statistiche del team, come ad esempio l'Utente più affaticato al momento o il numero di utenti online.

4.2.2 Dashboard server

Analizziamo ora la seconda componente del progetto, Dashboard server.
L'applicativo, come discusso alla *sezione 4.2*, implementa un server realizzato attraverso *Node.js*.
In particolare, attraverso il tool da riga di comando *Node package manager* (Npm) [26], è possibile installare il framework *Express* [27].
Attraverso Express ci viene fornita una funzionalità di Routing, cioè a fronte di una richiesta da parte di un client su un endpoint viene determinata la risposta del server.
Il Routing è definito da un URI e da un metodo HTTP GET e/o POST. Dipendentemente dal tipo di richiesta e dall'uri richiamato si scatenerà una determinata reazione.

4.2.2.1 Interfacciamento con database

Come discusso al *punto 3.3.1*, per una maggiore flessibilità si è optato per una soluzione che faccia uso di database non relazionali, in particolare utilizzando MongoDB.
Attraverso l'interfaccia *MongoClient* [28], fornendo le opportune configurazioni, è possibile connettersi al database MongoDB ed effettuare delle query sulle collezioni (definite al *punto 3.3.1.1*).

4.2.2.2 Endpoints

Attraverso questi endpoints, Dashboard server riceve richieste HTTP dall'applicativo Dashboard e interroga coerentemente il database MongoDB al fine di restituire un risultato (*figura 4.1*):

- ***login***
  Argomenti: username, password
  In caso di successo restituisce l'identificativo dell'organizzazione di cui il Team manager fa parte.
- ***teams***
  Argomenti: organizationId
  Restituisce una lista di Team collegati all'identificativo dell'organizzazione.
- ***teamSignals***
  Argomenti: organizationId, teamId, start, end
  Per ogni Utente all'interno di un Team restituisce i dati legati al benessere cognitivo con un timestamp compreso tra start e end.
- ***usersByOrganization***
  Argomenti: organizationId
  Restituisce tutti gli Utenti facenti parte dell'organizzazione.
- ***addUserToTeam***
  Argomenti: userId, organizationId, teamId
  Modifica i dati di un Utente collegandolo ad un Team della sua organizzazione.
- ***removeUserFromTeam***
  Argomenti: userId, organizationId, teamId
  Modifica i dati di un Utente rimuovendo il suo collegamento ad un Team della sua organizzazione.
- ***addTeam***
  Argomenti: organizationId, teamId
  Crea un nuovo team all'interno di un'organizzazione.
- ***deleteTeam***
  Argomenti: organizationId, teamId
  Elimina un Team all'interno di un'organizzazione.

- ***addUser***
  Argomenti: organizationId, userId
  Crea un nuovo Utente all'interno di un'organizzazione.

- ***deleteUser***
  Argomenti: organizationId, userId
  Elimina un Utente da un'organizzazione.

Capitolo 5

Validazione dell'applicativo Dashboard real-time

Nell'ultimo capitolo di questo elaborato andremo a validare quanto da me sviluppato durante l'esperienza in azienda: l'applicativo Dashboard real-time.
Per prima cosa parleremo del *simulatore di carico cognitivo*, un applicativo sviluppato al fine di testare quanto creato (oltre che progetti di natura simile a NeuroFrame).
Una volta compreso come il simulatore si inserisca nel progetto, andremo a *validare i sistemi implementati*, con viste sul Simulatore e sulla Dashboard in funzione.

5.1 Sviluppo di un simulatore di carico cognitivo

Come si evince da quanto detto fin ora, *il sistema Dashboard per funzionare necessita di interagire con il Database.*

I dati recuperati dal Database però vengono scritti dalla piattaforma Atlas che, a sua volta, li ha ricevuti dal sistema per l'Acquisizione di segnali.
Al momento della stesura di questo elaborato però *l'algoritmo MindPulse* (all'interno di Atlas) *risulta ancora in fase di perfezionamento e l'applicativo di Acquisizione in fase di sviluppo.*

Per verificare dunque il corretto funzionamento della Dashboard, è nata l'idea di sviluppare un **simulatore di carico cognitivo** *che vada a sostituirsi alla piattaforma Atlas nella scrittura di dati sul database MongoDB.*

5.1.1 Ruolo del simulatore nel sistema NeuroFrame

Il simulatore rende possibile ottenere continuamente nuovi dati sul Database così che la Dashboard possa mostrarli seguendo il suo normale funzionamento.
La *figura 5.1* riprende quanto visto precedentemente nella *figura 3.1*, escludendo però Atlas ed App di acquisizione e sostituendoli con il simulatore.

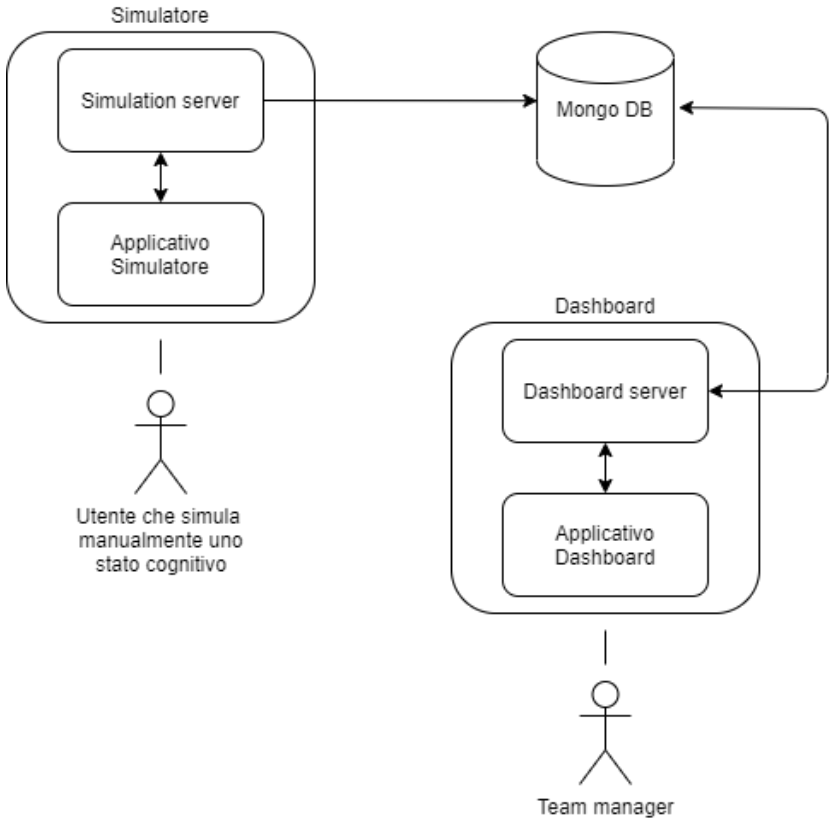

Figura 5.1: Architettura di NeuroFrame con simulatore di carico cognitivo

Data la volontà di Vibre di creare altri progetti (che sfruttano gli algoritmi della piattaforma) la cui architettura segua un'impostazione molto similare a NeuroFrame, *il simulatore è stato pensato con l'idea di essere riutilizzabile in altri contesti.*

Per correttezza infatti, *l'applicativo andrebbe visto come uno scrittore di dati su database a cadenza periodica*, piuttosto che come un simulatore prettamente legato a NeuroFrame; Il progetto prende dunque il nome di **Simulatore di Piattaforma**, simulando l'output su database di quest'ultima.

Se normalmente la Piattaforma Atlas elaborerebbe un input biometrico al fine di fornire metriche da utilizzare, *l'utente del Simulatore invece manipola manualmente il valore di tali metriche da inviare al database*, simulando il variare dei risultati ottenuti dagli algoritmi della piattaforma.

Al fine di rendere il Simulatore utilizzabile in più progetti, *l'utente potrà creare delle configurazioni che specifichino quali dati inviare, in che collezioni e in quale database.*

5.1.2 Flusso d' informazione in relazione agli altri componenti

Vediamo ora come il simulatore comunica con le componenti di NeuroFrame.
Il flusso d'informazione lato Dashboard, essendo esso un sottosistema che comunica principalmente con il Database, rimane invariato rispetto a quanto descritto nella *sezione 3.3.*

Il Simulatore invece va a sostituire il flusso informativo lato Applicativo di acquisizione, con una comunicazione più breve dovuta al fatto che sono coinvolti meno sistemi ed i dati non vengono elaborati dall'algoritmo MindPulse, ma direttamente decisi dall'utente che utilizza il Simulatore.
La logica alla base della comunicazione rimane invariata, inviando dati a MongoDB secondo una cadenza periodica.

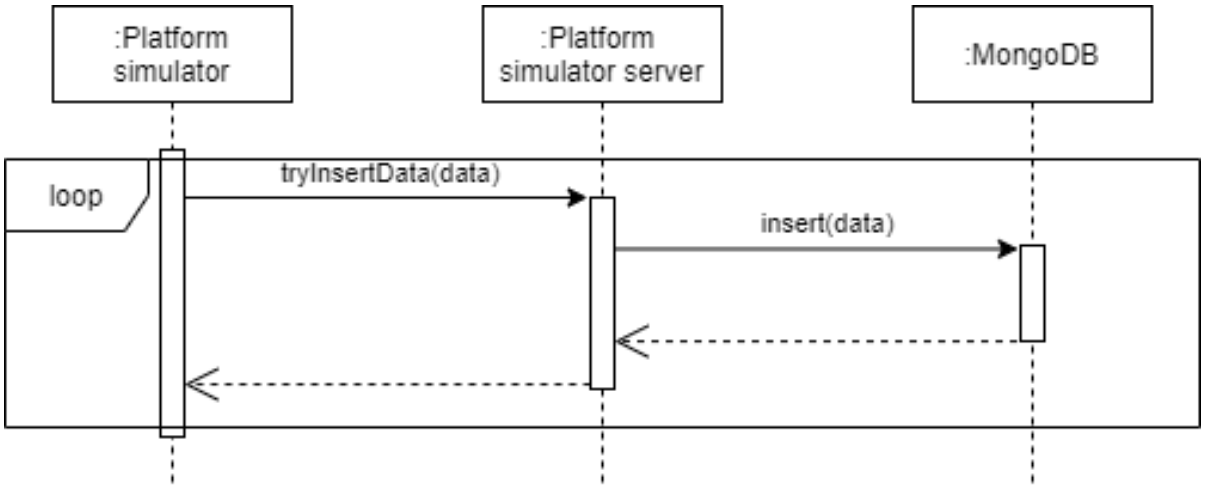

Figura 5.2: Diagramma di sequenza rappresentante il flusso informativo dal Simulatore a MongoDB

5.1.3 Soluzioni tecnologiche utilizzate

Analogamente al sistema Dashboard, il Simulatore è composto da due applicativi:

- *Un server realizzato tramite Node.js, con il supporto del framework Express (vedi sezione 4.2.2)*
- *L'applicativo con cui l'utente interagisce, sviluppato anch'esso tramite Node.js con l'ausilio del framework React (vedi sezione 4.1.1)*

Differentemente dall'applicativo Dashboard *non è stato necessario l'utilizzo di Redux* (*sezione 4.1.2*) in quanto l'applicativo risulta di essere di dimensioni molto contenute, rendendo gli strumenti offerti da React più che sufficienti per lo scopo.

Al fine di salvare delle configurazioni da riutilizzare in futuro, è stato implementato un *salvataggio di file locali* contenenti tutti i parametri necessari al fine di ripristinare tale configurazione; tali file contengono una *struttura in formato JSON* con il setting creato dall'utente.

5.2 Validazione dei sistemi implementati

In questa sezione conclusiva vedremo i sistemi implementati in funzione, esplicandone il funzionamento durante l'esecuzione concreta degli applicativi.

5.2.1 Simulatore in funzione

Partiamo dunque dal Simulatore di Piattaforma; esso, non essendo un applicativo destinato ad un uso commerciale ma bensì pensato solamente per un utilizzo aziendale in ambito di testing, non presentava dei requisiti stringenti e dispone di una GUI molto semplice e minimalista.
A schermo compare dunque solo l'indispensabile al funzionamento dell'applicativo.

Dipendentemente da se si dispone già o meno di un file di configurazione, nella **schermata iniziale** (*figura 5.3*), troviamo alcune strade che l'utente può intraprendere:

- **Creazione di una nuova configurazione**
  L'utente viene portato in una pagina atta alla creazione guidata di un nuovo file di configurazione.
- **Modifica di una configurazione esistente**
  Scegliendo questa opzione l'utente potrà caricare un file di configurazione presente nel suo dispositivo e modificarlo.
- **Inizio della Simulazione tramite una configurazione esistente**
  Analogamente al punto precedente, viene scelto un file di configurazione e si viene portati alla schermata da dove avviare la simulazione e controllarne i parametri.

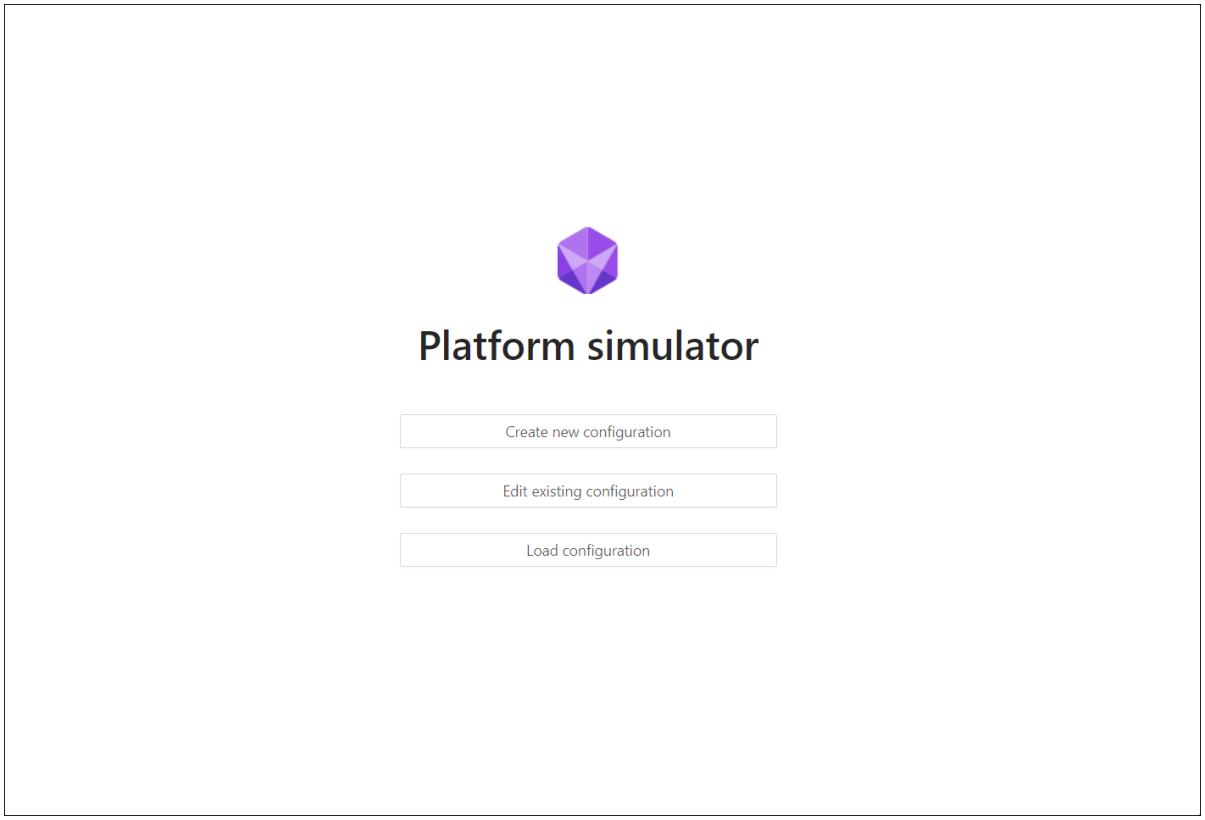

Figura 5.3: Schermata iniziale del Simulatore di Piattaforma

Seguendo una tra le prime due scelte dunque si viene portati alla **schermata di modifica di una configurazione** (*figura 5.4*).

Nel caso si stia creando una nuova configurazione i campi risulteranno ovviamente vuoti mentre, nel caso si stia modificando un file, saranno caricati i valori precedentemente salvati.

Partendo dalla struttura della collezione *Signals* (*sezione 3.3.1.1*) e con il presupposto che progetti futuri seguano quest'impostazione per salvare i dati elaborati, possiamo notare che la struttura di ogni entry sia astraibile in questo modo:

- *Chiave primaria composta*
  I campi la cui unione rende l'entry univoca all'interno della collezione; nonostante non tutti i campi siano prevedibili a priori, sappiamo con certezza che a comporre la chiavi vi saranno i timestamp *start* ed end.

- *Data*
  In questo array verranno inseriti progressivamente i dati elaborati; ogni entry di questo array contiene una struttura dati.
  Analogamente al caso precedente non possiamo sapere in anticipo quali siano i campi che verranno salvati ma sappiamo che in qualsivoglia struttura dati dovrà essere salvato il timestamp relativo all'istante temporale in cui quel dato è stato acquisito.

Come conseguenza di queste considerazioni sono state definite tre macroaree di input all'interno della schermata:

- **Parametri MongoDB**
  È qui che l'utente inserisce i dati obbligatori al fine di interfacciarsi con il database MongoDB.
  Sono necessari:
  - *L'uri, dove reperire il DBMS*
  - *Il nome del database di riferimento all'interno del DBMS*
  - *Il nome della collezione in cui vogliamo scrivere dati*

- **Chiavi**
  L'utente inserisce tutte le chiavi necessarie a creare la chiave primaria composita che identifichi univocamente ogni "chunk" (entry) nella collezione (è richiesto almeno l' inserimento di una chiave).
  Per ogni chiave vanno specificati nome ed valore assunto.

- **Variabili**
  Analogamente al punto precedente, in questa area vanno inserite tutte le variabili da manipolare durante la simulazione e che andranno a comporre la struttura dati di ogni entry dell'array *Data*.
  Per ogni variabile inserita va definito un massimo ed un minimo (oltre che il suo nome).

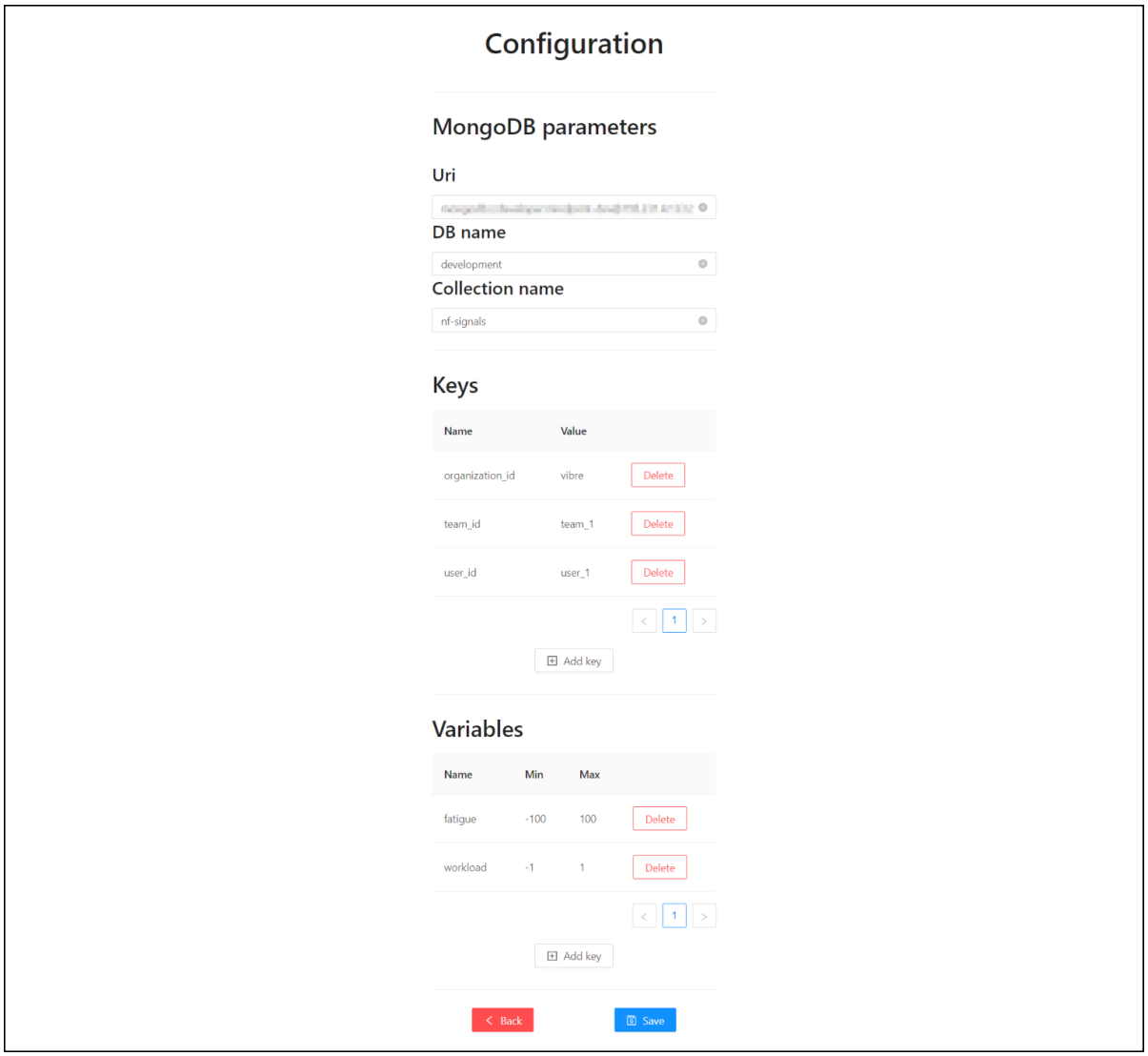

Figura 5.4: Schermata di modifica/creazione di una configurazione con dati relativi ad una simulazione nel sistema NeuroFrame

L'ultima schermata disponibile infine è quella in cui tutte e tre le scelte iniziali possibili

vanno a culminare: **la schermata di simulazione** (*figura 5.5*).

Qui ritroviamo le tre sezioni descritte nella precedente schermata; se i parametri di connessione con il database e le chiavi non consentono interazione, nella sezione contenente le variabili ritroviamo una sottosezione per ognuno dei campi creati.
È possibile modificare il valore di ogni variabile (all'interno dell'intervallo definito da *max* e *min*) attraverso uno slider o tramite un componente apposito che permette sia di aumentarlo/diminuirlo a piccoli step sia di specificare direttamente il valore in forma numerica.

In cima alla pagina inoltre sono presenti un bottone per tornare alla schermata iniziale ed uno per iniziare/interrompere l'invio dei dati.
L'applicativo gestirà in automatico i timestamp descritti sopra in quanto campi obbligatori nella struttura di ogni chunk.

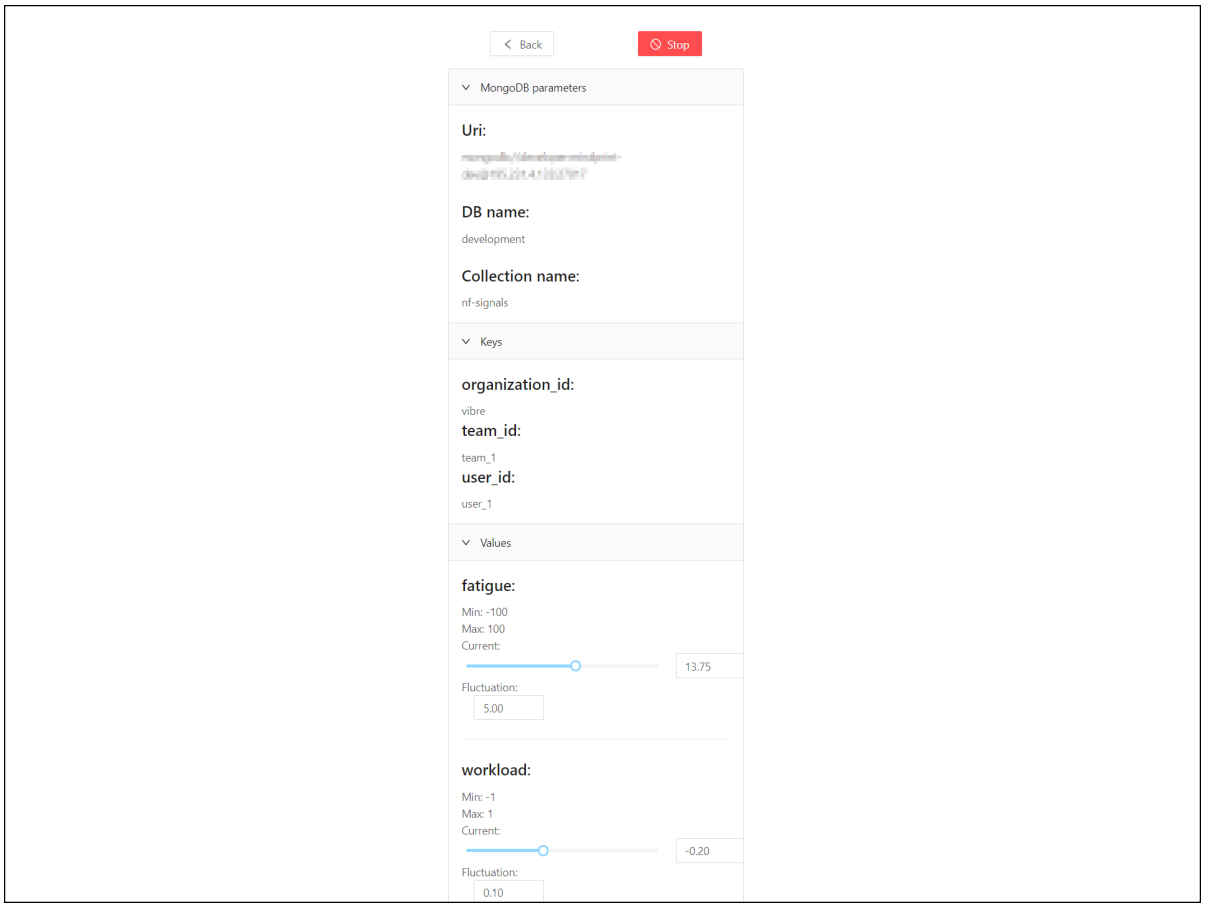

Figura 5.5: Schermata di simulazione, con invio dati correntemente in corso

5.2.2 Sistema Dashboard in funzione

Siamo finalmente giunti a visionare la Dashboard sviluppata come sottosistema di NeuroFrame.

La schermata di **Login** (*figura 5.6*) riprende sostanzialmente in toto quanto visto nel relativo wireframe (*figura 2.10*); qui troviamo un form di inserimento per le credenziali del Team manager che userà l'applicativo, nonchè il logo di Vibre.

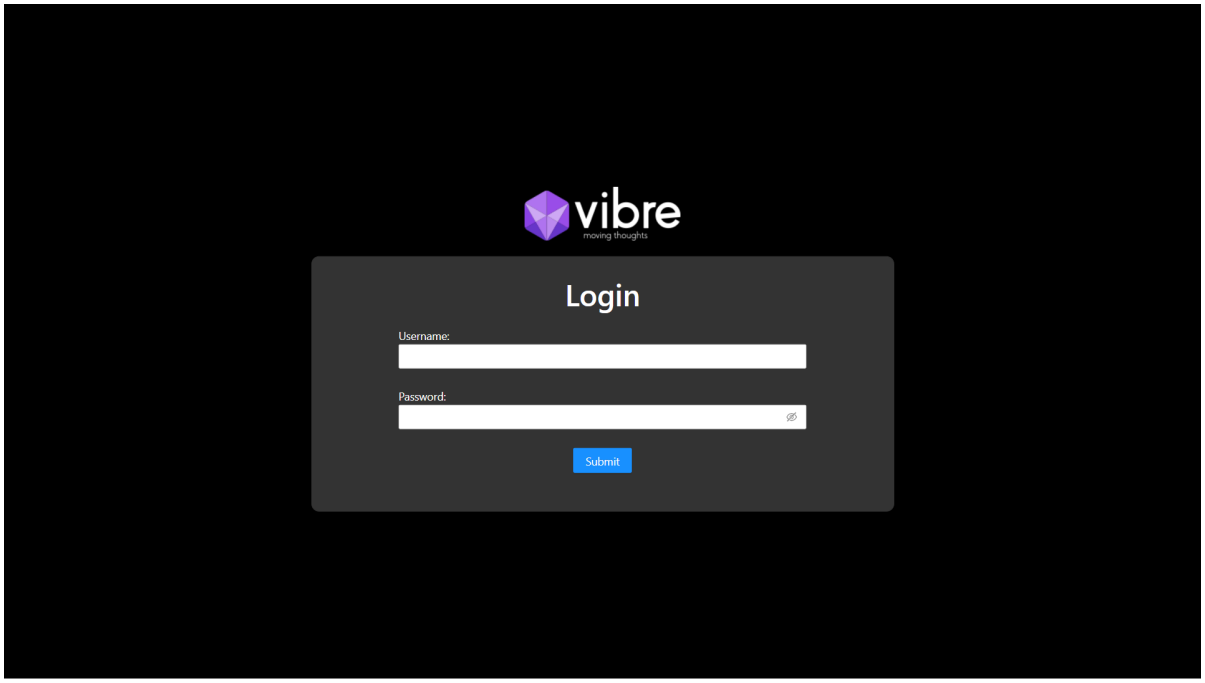

Figura 5.6: Schermata di login del sistema Dashboard

Successivamente all'autenticazione del Team manager comparirà la **schermata di selezione del Team** (*figura 5.7*); qui saranno elencati i Team presenti all'interno dell'azienda e, per ognuno di essi, verrà mostrato l'identificativo dei suoi componenti.

Nella barra in alto inoltre troviamo due pulsanti. Attraverso la pressione del primo pulsante si potrà accedere alla **gestione dei Team** (*figura 5.8*) mentre, premendo il secondo, si accederà alla **gestione degli utenti** (*figura 5.9*).

Figura 5.7: Schermata di selezione del Team

Nella schermata di gestione dei Team sarà possibile eseguire due operazioni:

- *Creare nuovi Team*, rispettando il numero massimo imposto dalla licenza acquistata dall'organizzazione.
- *Eliminare Team esistenti*, previa rimozione di ogni Utente dal Team attraverso la schermata apposita che verrà affrontata successivamente.

Similmente, nella schermata dedicata alla gestione degli Utenti sarà possibile:

- *Creare nuovi Utenti*
- *Eliminare Utenti che non facciano parte di un Team*
- *Visionare un elenco degli Utenti correntemente appartenenti ad un Team; per ogni Utente verrà mostrato il relativo Team di appartenenza.*

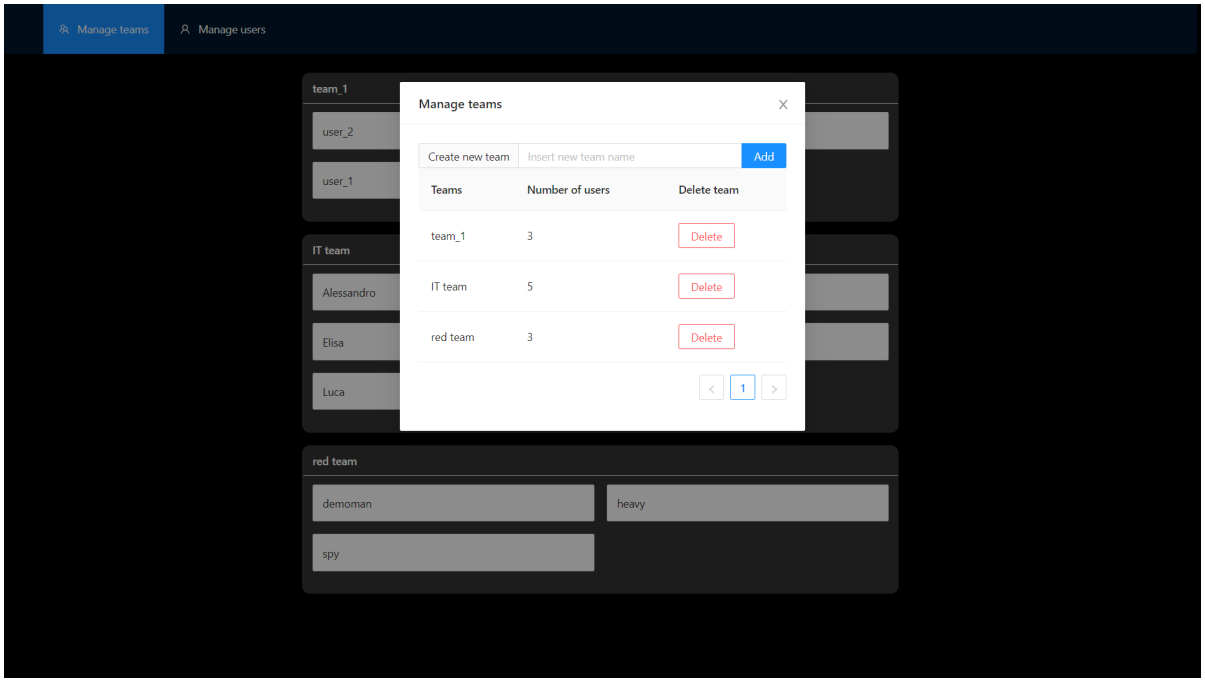

Figura 5.8: Schermata di gestione dei Team

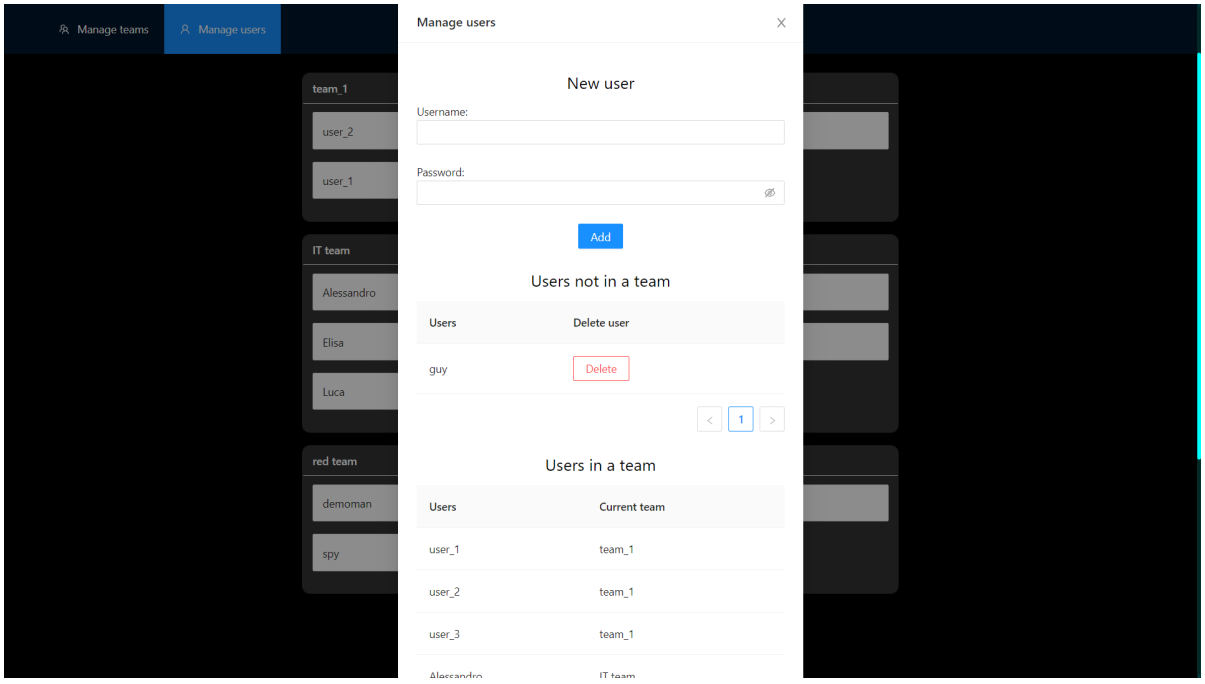

Figura 5.9: Schermata di gestione degli Utenti

Una volta scelto un Team nella schermata di selezione di quest'ultimi, si viene portati nella schermata core dell'applicativo: **la schermata della dashboard di monitoring del Team** (*figura 5.10*).

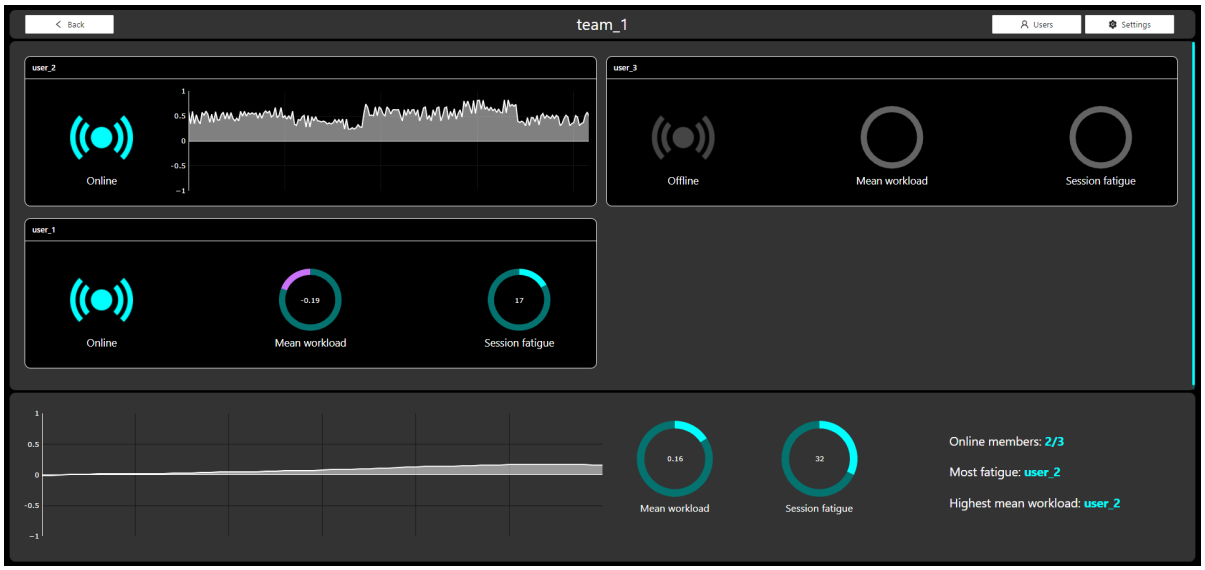

Figura 5.10: Schermata della dashboard di monitoring del Team

Seguendo le linee guida del wireframe nella *figura 2.11* precedentemente discussa, ritroviamo le tre macro aree pianificate:

- *Impostazioni*
  Il risultato finale differisce leggermente da quanto rappresentato nel wireframe.
  All'interno della barra delle impostazioni posizionata in cima alla pagina il logo di Vibre e quello dell'organizzazione a cui appartiene il Team leader che sta utilizzando l'applicativo sono assenti; tale assenza è da imputarsi a motivazioni legate allo spazio disponibile a schermo (ricordiamo che questa schermata deve occupare la totale altezza dello schermo senza però poter scorrere in verticale).
  Sulla sinistra troviamo un bottone per tornare alla schermata di selezione del Team mentre, sulla destra, troviamo due pulsanti per le impostazioni che verranno discussi a breve:

  – **Schermata della gestione degli Utenti del Team** (*figura 5.11*)
  – **Schermata delle impostazioni di visualizzazione dati** (*figura 5.12*)

- *Area singoli componenti*
  In quest'area è presente una sezione dedicata ad ogni componente del Team.
  Essendo il numero di Utenti non prevedibile a priori questa è l'unica sezione della schermata che può scorrere in verticale.
  Per ogni Utente suddividiamo gli indicatori in due aree:

  - *Indicatore di connessione*
    Descrive lo stato di connessione dell'Utente in questione.
    Al momento può assumere solo uno stato booleano (online o offline) ma, in versioni future, è prevista la possibilità di visualizzare la qualità del segnale inviato dall'Utente durante lo streaming dati.
  - *Indicatori delle metriche neurali*
    Qui vengono mostrate le metriche calcolate dall'algoritmo MindPulse (o fornite dal Simulatore).

    In figura vediamo, per lo *user 1*, la visualizzazione di *carico cognitivo medio* e *affativamento medio* tramite grafici a torta; data la possibilità di queste metriche di assumere anche valori negativi, i grafici si riempiono in senso orario (con colore azzurro) all'aumentare in valori superiori allo zero mentre, per valori negativi, si riempiono in senso antiorario (con un colore viola chiaro).
    Valori positivi simboleggiano rispettivamente un utilizzo di risorse mentali (per il carico cognitivo) ed un affaticamento mentale dell'individuo; di contro quando queste metriche assumono valori negativi l'utente sarà meno concentrato e più rilassato.
    Superati dei valori critici, i grafici assumono una colorazione rossa ed il sistema avvisa il Team manager che, ad esempio, consiglierà all'Utente stanco una pausa.

    Per lo *user 2* troviamo invece una diversa rappresentazione dei dati; l'affaticamento non viene mostrato ed inoltre, attraverso un grafico cartesiano, non viene mostrata la media del carico cognitivo bensì *i valori che esso assume nel tempo*(è su questi valori difatti che la media viene calcolata).
    Per cambiare la modalità di visualizzazione delle metriche neurali sarà sufficiente un click del Team manager nello spazio dell'Utente d'interesse.

- *Area del Team*
  Qui vengono mostrate a schermo delle metriche che riassumono la condizione generale del Team.
  A sinistra troviamo un grafico cartesiano che descrive l'andamento *della media* di tutti i carichi cognitivi nel tempo.

Subito a destra ritroviamo i grafici a torta prima affrontati, rappresentanti le medie attuali rispettivamente di carico cognitivo e di affaticamento mentale.
Infine è presente una sezione con delle info quali numero di Utenti online, quale sia l'Utente più affaticato al momento così come l'utente con carico cognitivo maggiore.

Riprendendo il discorso che concerne le schermate riguardanti le impostazioni, analizziamo in primo luogo quella riguardante la gestione degli Utenti del Team (*figura 5.11*).
Il Team manager in questa schermata può rimuovere Utenti dal Team o aggiungerne di nuovi che non siano già appartenenti ad un Team.
Il sistema impedirà di aggiungere Utenti nel caso si sia raggiunto il numero massimo di Utenti per Team consentito dalla licenza acquistata dall'organizzazione.

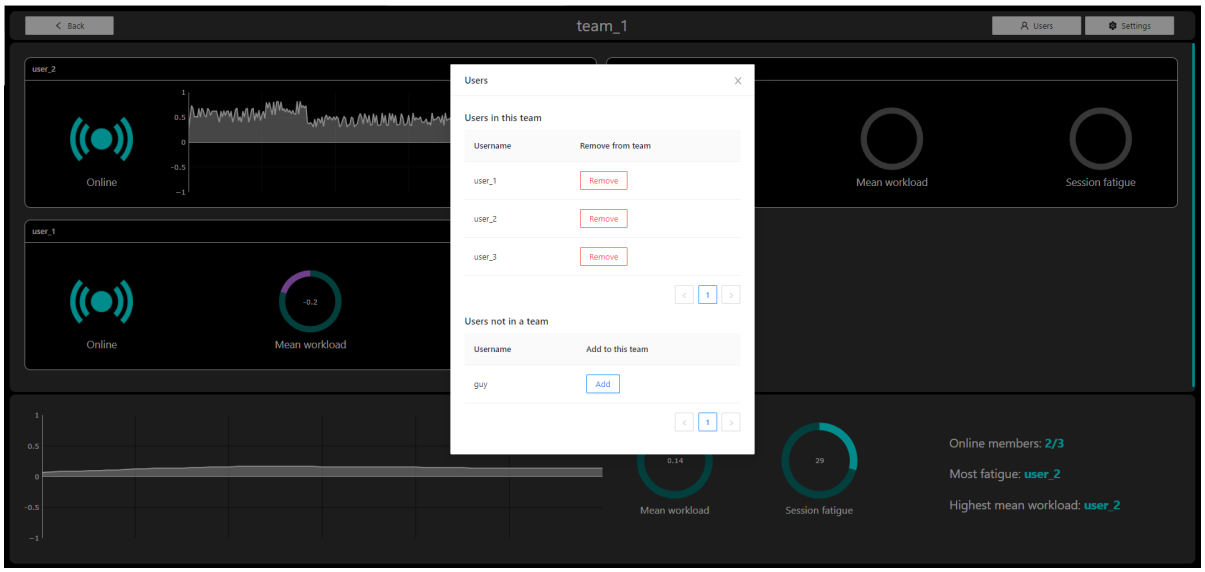

Figura 5.11: Schermata della gestione degli Utenti del Team

La seconda schermata di opzioni invece riguarda le impostazioni di visualizzazione dati(*figura 5.12*).
I cambiamenti in questa sezione saranno relativi solamente alla modalità di visualizzazione dati del Team manager che sta utilizzando l'applicativo e non si ripercuote su altri Team manager della stessa organizzazione.

In questa prima versione sarà possibile agire su due parametri:

- *Modificare la finestra temporale per il carico cognitivo*
  Attraverso questo parametro verrà impostato lo span temporale rappresentato dai grafici cartesiani rappresentanti l'evoluzione temporale del carico cognitivo nel tempo.
  Ad esempio, impostando 60 secondi, i valori più vecchi di tale parametro non verrebbero più visualizzati nei grafici cartesiani e, conseguentemente, non verrebbero nemmeno considerati per il calcolo del carico cognitivo medio.
  È stato imposto un limite superiore di 300 secondi per evitare un possibile rallentamento del sistema dato dalle risorse limitate del browser.

- *Modificare il valore massimo assoluto dell'affaticamento mentale*
  Se la metrica restituita da MindPulse riguardo il carico cognitivo è inclusa in un range che va da -1 a 1, l'affaticamento mentale invece non dispone né di un limite inferiore né di un limite inferiore.
  Studi e test al riguardo hanno verificato che in situazioni ordinarie tale valore non cresca oltre una certa soglia, fissata di default a -100 e 100 nella Dashboard.
  Un limite superiore ed inferiore è necessario al funzionamento sopra descritto dei grafici a torta; nel caso compaiano valori che superino i limiti, essi verranno comunque mostrati in forma numerica al centro del grafico che, però, non si riempirà oltre la capienza massima, sia positiva che negativa.
  Viene però fornita la possibilità di variare il valore assoluto di tale range nel caso durante l'utilizzo compaiano frequentemente valori fuori anomali fuori scala.

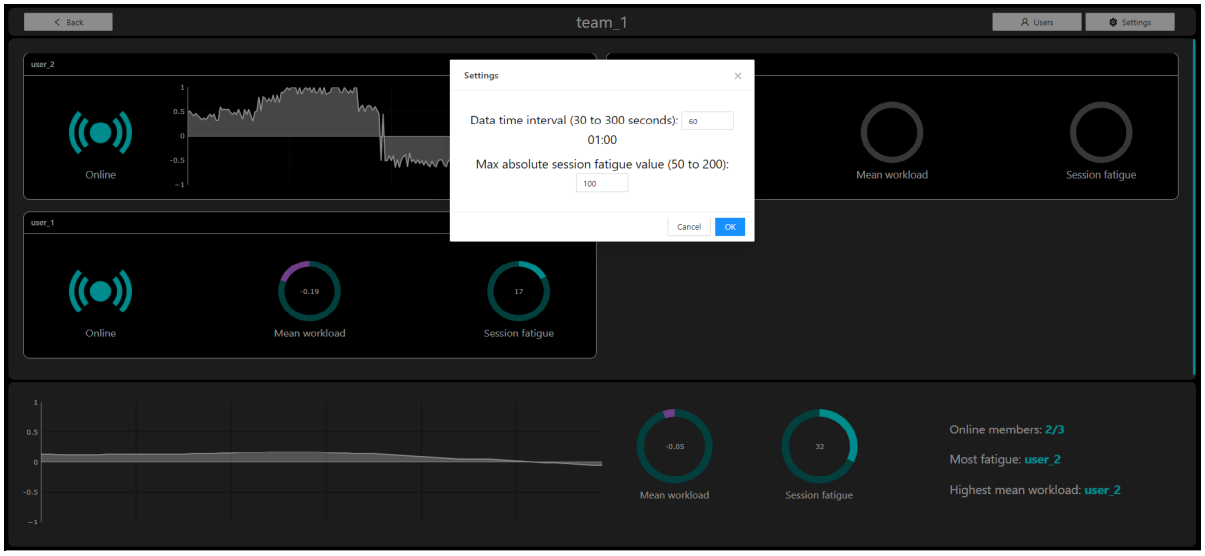

Figura 5.12: Schermata delle impostazioni di visualizzazione dati

Considerazioni finali

Data la natura molto ampia di questo progetto, saranno necessari ulteriori sforzi per ottenere una versione di NeuroFrame funzionante in ogni sua parte.

D'altro canto, grazie ad un meticoloso processo di modellazione perpetuato dall'intera sezione IT di Vibre, i sottosistemi componenti il macrosistema NeuroFrame godono di un basso grado di dipendenza e ciò permette una valutazione ed uno sviluppo di ciascuno di essi.

Dunque, ritenendo che il processo di ingegnerizzazione sia stato da noi svolto al meglio delle nostre capacità cercando di seguire le best practices della disciplina, vorrei trarre le conclusioni sul focus implementativo della mia esperienza: il sottosistema Dashboard.

Riprendendo l'analisi dei requisiti del sistema, argomento della *sezione 2.1*, tutti gli obbiettivi con priorità massima (Must have) concernenti la Dashboard sono stati raggiunti, ottenendo una prima versione funzionante dell'applicativo.
Non sono comunque stati trascurati i requisiti con priorità minore da implementare in versioni future, creando un'architettura già predisposta a realizzare tali feature.

Sviluppare il Simulatore, oltre a permettere l'effettivo accertamento dell'operatività della Dasbhoard, è stata un'esperienza molto formativa, permettendomi di mettere in atto soluzioni volte alla creazione di un tool general purpose che possa essere di valore per l'azienda anche in contesti che esulano dal mio progetto di tesi.

Il prossimo step nel futuro di questo progetto vede la realizzazione dell'Applicativo di acquisizione secondo quanto modellato in questo elaborato, oltre che il perfezionamento definitivo del già ottimo algoritmo MindPulse.

Credo molto nell'innovazione che questo progetto può apportare nel mondo del lavoro e spero che esso possa aiutare quanti più lavoratori possibili nelle sfide che affrontano ogni giorno.

Bibliografia

[1] Roberto Bernabò. *Chi sono i lavoratori della conoscenza - kwnoledge workers?* URL: https://www.socialdigitalknowledge.com/2019/11/21/chi-sono-i-lavoratori-della-conoscenza-knowledge-workers/#:~:text=L'espressione%20%E2%80%9Clavoratori%20della%20conoscenza,conoscenza%20operano%20sui%20processi%20immateriali..

[2] Davorka Filipusic O. Geoffrey Okogbaa Richard L. Shell. *[Applied Ergonomics 1994 Volume 25 Numero 6 pag 356-357] On the investigation of the neurophysiological correlates of knowledge worker mental fatigue using the EEG signal [Mental fatigue].* URL: https://scihub.wikicn.top/https://www.sciencedirect.com/science/article/abs/pii/000368709490054X.

[3] A. Lasalvia. *Occupational stress and job burnout in mental health.* URL: https://www.cambridge.org/core/services/aop-cambridge-core/content/view/B6F96F3F103D77DA46F79905DB725C02/S2045796011000576a.pdf/occupational-stress-and-job-burnout-in-mental-health.pdf.

[4] Prof. dr. Wilmar Schaufeli. *Burnout in Europe. Relations with National Economy, Governance and Culture.* URL: https://www.wilmarschaufeli.nl/publications/Schaufeli/500.pdf.

[5] *What is a flow state and what are its benefits?* URL: https://www.headspace.com/articles/flow-state.

[6] *Interfaccia neurale.* URL: https://it.wikipedia.org/wiki/Interfaccia_neurale.

[7] G. Di Flumeri P. Aricò G. Borghini. *[pag 2-4]Passive BCI beyond the lab: current trends and future directions.* URL: https://scihub.wikicn.top/https://iopscience.iop.org/article/10.1088/1361-6579/aad57e.

[8] *Muse headbands.* URL: https://choosemuse.com/.

[9] *Muse 2 review – the device to help you achieve calm through meditation.* URL: https://www.techguide.com.au/reviews/gadgets-reviews/muse-2-review-device-help-achieve-calm-meditation/.

[10] *MoSCoW Prioritization.* URL: https://www.productplan.com/glossary/moscow-prioritization/#:~:text=The%20method%20is%20commonly%20used,not%20have%20at%20this%20time..

[11] *An introduction to Agile modeling.* URL: http://www.agilemodeling.com/essays/introductionToAM.htm.

[12] Marteen Van Steen Andrew S. Tanenbaum. *[Sistemi distribuiti, seconda edizione - Sezione 1.1]Definizione di sistema distribuito.* URL: https://books.google.it/books?hl=it&lr=&id=uhrblChZ5_wC&oi=fnd&pg=PR15&dq=sistemi+distribuiti&ots=1JSYdXRG8f&sig=Djek_Bk0ySRPJ8gu8OcrB4vLrKU#v=onepage&q=sistemi%20distribuiti&f=false.

[13] *What are microservices?* URL: https://microservices.io/.

[14] *Cos'è SaaS?* URL: https://www.salesforce.com/it/learning-centre/tech/saas/.

[15] *What Is MongoDB?* URL: https://www.mongodb.com/what-is-mongodb.

[16] *Introduzione a JSX.* URL: https://it.reactjs.org/docs/introducing-jsx.html.

[17] *Componenti e props.* URL: https://it.reactjs.org/docs/components-and-props.html.

[18] *State e lifecycle.* URL: https://it.reactjs.org/docs/state-and-lifecycle.html.

[19] *Redux, A Predictable State Container for JS Apps.* URL: https://redux.js.org/.

[20] *Store.* URL: https://redux.js.org/basics/store.

[21] *Reducers.* URL: https://redux.js.org/basics/reducers.

[22] *Actions.* URL: https://redux.js.org/basics/actions.

[23] *Redux toolkit.* URL: https://redux-toolkit.js.org/.

[24] *Thinking in thunks.* URL: https://redux-toolkit.js.org/tutorials/advanced-tutorial#thinking-in-thunks.

[25] *Ant design.* URL: https://ant.design/docs/react/introduce.

[26] *Node package manager - Npm.* URL: https://docs.npmjs.com/about-npm/.

[27] *Express.* URL: https://expressjs.com/it/.

[28] *MongoClient.* URL: https://mongodb.github.io/node-mongodb-native/api-generated/mongoclient.html.

Ringraziamenti

Dedicato alla mia famiglia; è solo grazie ai vostri sforzi, i vostri sacrifici ed il vostro sostegno se oggi sono qui.
Grazie ai miei genitori per aver portato pazienza ed essermi stati accanto anche nei momenti in cui non me lo meritavo, vi voglio un mondo di bene.

Ad i miei amici, a tutti gli anni trascorsi insieme, le avventure, le risate...
Grazie di aver portato luce nella mia vita anche nei momenti più bui, siete i miei fratelli e le mie sorelle.

Alla mia ragazza, alla tua dolcezza, al tuo cuore d'oro, al tuo sorriso e alla tua positività.
Grazie per aver portato l'amore nella mia vita.

Ai miei compagni di corso, a tutte le gioie ed i dolori, alle giornate trascorse insieme sui libri facendoci forza a vicenda.
Grazie per il vostro supporto che mi ha spinto a resistere anche quando avevo solo voglia di mollare.

Ai miei colleghi in Vibre, grazie per avermi accolto così gentilmente ed avermi offerto questa incredibile possibilità che mi ha fatto crescere come professionista ma soprattutto come persona.
Grazie in modo particolare a Luca e Marco, per avermi accompagnato in questo percorso; senza di voi questa tesi non esisterebbe.

Ai professori, ai loro insegnamenti, alla loro disponibilità.
Grazie al professor Ricci, per avermi aiutato e supervisionato in questo percorso di tesi e per avermi mostrato tramite i suoi insegnamenti realtà che non aveno nemmeno mai preso in considerazione.

Alle persone che, per un motivo o per un altro, non sono più nella mia vita, perchè è anche grazie a voi che oggi sono quello che sono.
A te nonno, mi manchi tanto.