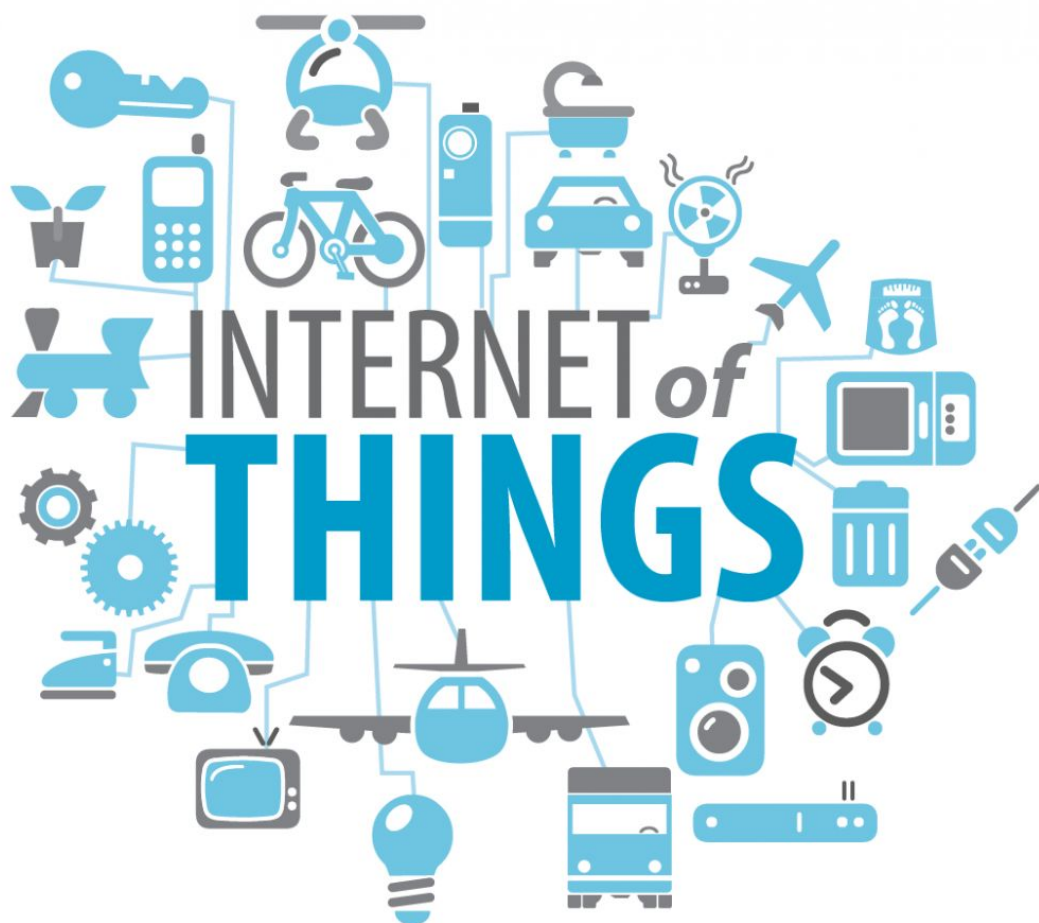


Relazione sul Progetto di Reti 2

Sperimentazioni



SmartBench

Una panca piana intelligente

Andrea Ierardi, 20018785

Marco Rizzi, 20019238

Sommario

Introduzione	2
Specifiche funzionali da soddisfare	2
Analisi della tecnologia utilizzata	3
Scelta dell'approccio	4
Architettura del Software	6
Descrizione dell'implementazione effettuata	6
Descrizione dell'Interfaccia Utente	11
Interfaccia di Configurazione	11
Pagina di login	12
Interfaccia Grafica per l'utente	13
Validazione del software strutturato	15
Bibliografia	16

Introduzione

Lo scopo di questa relazione è quella di illustrare lo sviluppo, il funzionamento e la realizzazione di una panca piana intelligente che monitora tramite l'utilizzo di appositi sensori lo stato, la performance e la salute dell'atleta durante l'allenamento, e si adatta per aiutarlo a completarne l'esecuzione in modo sicuro.

In particolare, la SmartBench legge ed elabora i dati che riceve da tre sensori diversi: pesi, bilanciere e bracciale. I primi due monitorano rispettivamente la variazione nel peso sollevato e lo svolgimento di una ripetizione (sollevamento e abbassamento del bilanciere), mentre l'ultimo monitora la frequenza e il battito cardiaco dell'atleta. Oltre a questo, l'atleta può riporre il bilanciere in qualsiasi momento per effettuare una pausa. In questo caso vi è un sensore che controlla che il bilanciere sia stato riposto correttamente.

Un utente, per poter utilizzare la SmartBench, deve essere registrato alla palestra, e può essere un atleta o un trainer. Nel secondo caso, l'utente ha accesso, oltre al normale utilizzo della panca, anche ad alcune funzionalità extra, cioè creare e gestire dei piani di allenamento personalizzati per gli atleti che gestisce. La Smartbench deve poter comunicare verso la rete esterna, in quanto scrive il resoconto degli allenamenti su un Database posizionato in cloud.

Specifiche funzionali che devono essere soddisfatte

Prima di introdurre i requisiti specifici del progetto è doveroso un accenno ad alcuni requisiti generali:

- Il sistema deve essere installato in reti private senza dover intervenire con configurazioni che riguardano l'apparato di rete.
- L'utente deve avere accesso ai dispositivi da un qualunque punto della rete, sia dalla LAN in cui risiedono i dispositivi che dalla rete Internet pubblica;

A questo punto possiamo entrare nel dettaglio e andare a vedere i requisiti necessari per realizzare la SmartBench:

- La scheda integrata a disposizione è una Beaglebone provvista di 8 ingressi, collegati a dei sensori, e con sistema operativo Linux Debian-8.
- È previsto un sistema per permettere all'utente di interrompere l'allenamento senza doverlo completare per intero.
- E' previsto, inoltre, che l'utente possa in qualsiasi momento riposizionare il bilanciere nell'apposito supporto e terminare la serie corrente, senza però interrompere l'intero allenamento.
- Per garantire la sicurezza dell'utente durante l'allenamento, è previsto un sistema di monitoraggio del livello del bilanciere, il quale attiva un allarme quando l'altezza del bilanciere misurata rimane a un livello critico per troppo tempo.
- L'applicazione deve svolgere funzioni di log e utilizzare i dati ricavati per incrementare le funzionalità offerte all'utente;
- Dev'essere messa a disposizione un'interfaccia grafica (GUI) che permetta all'utente di accedere allo stato del sistema e svolgere un allenamento, vedere lo storico dei suoi allenamenti (o nel caso di un trainer, lo storico di allenamenti di altri atleti), e altre funzioni specifiche.

Analisi della tecnologia utilizzata

Uno dei protocolli maggiormente utilizzati nel nostro progetto per la comunicazione fra Client e SmartBench, Database e i sensori è stato il protocollo MQTT tramite l'uso di un message broker, Mosquitto. Un broker è un applicativo che funge da intermediario tra i vari oggetti, collegati ad esso in maniera asincrona utilizzando il protocollo di messaggistica MQTT di tipo publish-subscribe. Un broker può essere visto come una "bacheca", dove i mittenti, detti publisher, pubblicano sotto a uno degli argomenti della bacheca (detti topic). A questi topic sono "abbonati" degli altri componenti, detti subscriber, che ricevono solo i messaggi indirizzati ai topic a cui loro sono abbonati. Il broker inoltra ogni messaggio inviato da un publisher verso tutti i subscribers interessati a quel "tipo" (Topic) di messaggio. Ciascun messaggio è indirizzato verso un Topic (argomento), che possiamo considerare un po' come una categoria/sottocategoria a cui appartengono.

Come è facile intuire, quindi, non è necessario che un oggetto IoT interroghi ripetutamente il broker per sapere se ci sono nuovi messaggi da leggere: ogni subscriber viene notificato automaticamente dal broker qualora ci sia un nuovo messaggio da consegnargli. Inoltre, non si perdono mai messaggi sulla rete, in quanto il broker si occupa di ricevere e inoltrare i messaggi ai dispositivi interessati, tramite il meccanismo descritto precedentemente. Questo schema molto efficace è anche estremamente scalabile grazie al fatto che i publisher non devono conoscere quanti e quali siano i subscriber a cui mandare i messaggi e viceversa.

Per la configurazione iniziale dei valori costanti è stato utilizzato il protocollo HTTP attraverso l'architettura REST. Quest'ultima necessita dell'utilizzo di URL aventi una struttura ben definita per identificare univocamente una risorsa o un insieme di risorse, oltre all'utilizzo dei verbi HTTP specifici GET, POST, PUT, PATCH, DELETE per la modifica e il recupero di informazioni e OPTIONS, ecc per altri scopi. L'approccio architetturale REST è definito dai seguenti sei vincoli applicati ad una architettura, mentre si lascia libera l'implementazione dei singoli componenti:

1. **l'architettura è di tipo client-server.** Il client e il server sono separati, e ognuno di essi ha diverse funzionalità. Ad esempio, il client non deve memorizzare le informazioni, che sono memorizzate sul server. Viceversa, i server non si devono preoccupare dell'interfaccia grafica o dello stato dell'utente, e di conseguenza sono più semplici e maggiormente scalabili.
2. **La comunicazione è priva di stato (Stateless).** Ogni richiesta da ogni client contiene tutte le informazioni necessarie per richiedere il servizio, e lo stato della sessione è contenuto sul client.
3. **Caching delle risposte.** Una gestione ben fatta della cache può ridurre o parzialmente eliminare le comunicazioni client-server, migliorando scalabilità e performance.

4. **Sistema a strati.** Un client non può sapere se è connesso direttamente ad un server di livello più basso od intermedio. I server intermedi possono migliorare la scalabilità del sistema con load-balancing o con cache distribuite. Layer intermedi possono offrire inoltre l'implementazione di politiche di sicurezza.
5. **Codice trasferibile:** I server possono temporaneamente estendere o personalizzare le funzionalità del client trasferendo del codice eseguibile. Ad esempio questo può includere componenti compilati come Applet Java o linguaggi di scripting client side come JavaScript.
6. **Interfaccia uniforme:** Un'interfaccia uniforme tra client e server permette di semplificare e sviluppare separatamente l'architettura.

Nel nostro caso, l'HTTP server viene utilizzato per permettere al tecnico di configurare il tempo di esecuzione di una serie, il tempo di pausa tra una serie e un'altra, e il tempo di monitoraggio del livello del bilanciamento (ovvero quanti secondi possono passare senza che venga effettuata una ripetizione prima di mandare il segnale di allarme) scegliendo dei valori da un apposito menù a tendina. Inoltre vi è presente un bottone di reset, che resetta l'applicazione ai valori di default e reinizializza tutto il sistema, e un bottone di shutdown di emergenza che spegne completamente il sistema. Per essere riattivato dopo lo shutdown di emergenza, il sistema va riavviato da zero.

Descritta in maniera generale la tipologia di architettura adottata, si può scendere nei dettagli delle scelte implementative intraprese.

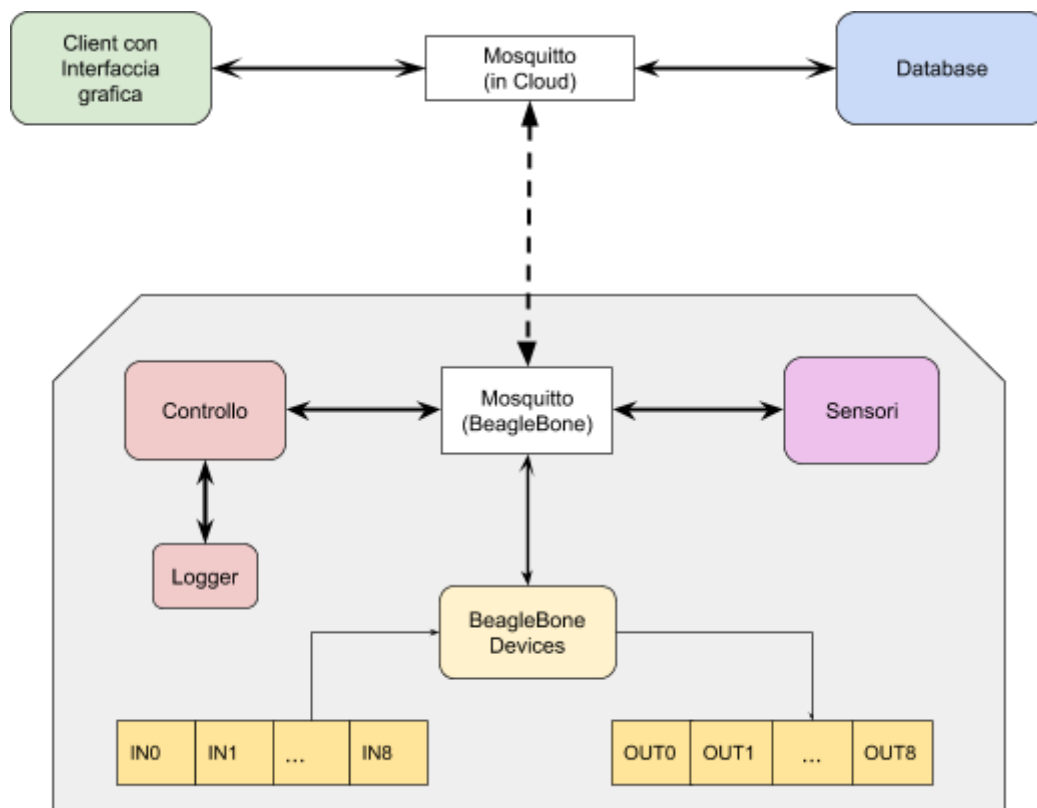
Scelta dell'approccio tecnologico

L'architettura utilizzata da noi è stata quella del message broker. Si è scelto di usare questo tipo di architettura in quanto si incastra perfettamente nel nostro progetto data la possibilità di rendere ogni componente del sistema modulare.

Per entrare nello specifico, nella nostra applicazione si è deciso di utilizzare non uno, ma due broker Mosquitto: uno posto sul cloud, e uno locale posto sulla Beaglebone. Il broker locale mette in comunicazione sensori e controllo della SmartBench, posti anch'essi sulla Beaglebone, mentre il broker in cloud permette di far comunicare tra loro controllo, client e database. Quest'ultimo, posto in cloud, contiene tutti gli allenamenti, i piani e gli utenti registrati alla palestra che utilizza la SmartBench, come visto nello schema qui sotto. Oltre al controllo e ai sensori, sulla Beaglebone si trova un terzo modulo separato, il Logger, che tiene traccia dei messaggi inviati e ricevuti dal controllo, e tutte le operazioni effettuate da quest'ultimo. Questa componente è separata dal controllo, in quanto se ci fosse un guasto su di essa non comporterebbe lo spegnimento totale dell'intera SmartBench, ma solo di quel modulo, e per evitare l'intasamento della memoria riservata al dispositivo con i messaggi del Logger.

Dato che i device IoT che utilizziamo dispongono di poca memoria, abbiamo la necessità che il traffico dati sia ridotto al minimo, per non sovraccaricare le suddette memorie. Ed è proprio per questo che utilizziamo il broker: esso permette infatti di non richiedere una costante connessione dei subscriber per verificare la ricezione dei messaggi, ma li notifica lui stesso, permettendo così ai subscriber di risparmiare energia tra una pubblicazione e l'altra.

Dopo attente revisioni, la nostra architettura si può riassumere nello schema seguente:



Architettura del Software

Descrizione dell'implementazione effettuata

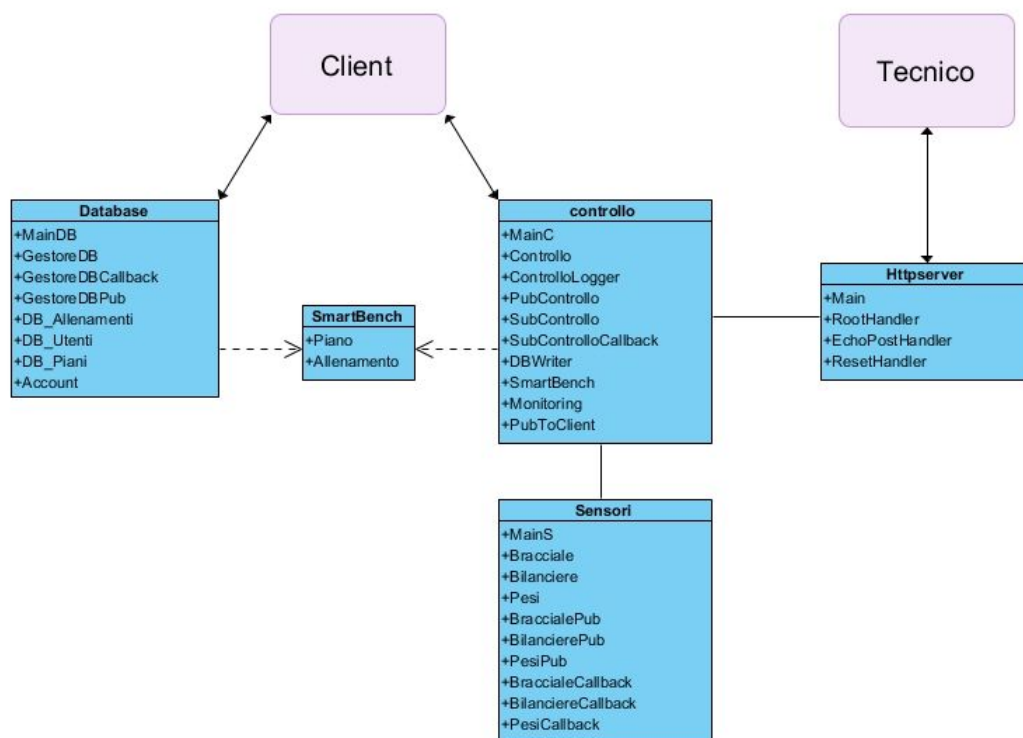
Prima di passare a descrivere le classi utilizzate, è necessario precisare che sono stati utilizzati Thread per gestire in maniera concorrente i vari task, garantendo una sincronizzazione tra di essi, così da avere un guadagno in termini di tempo ottimizzando lo sfruttamento delle risorse del sistema, ulteriormente favorita dall'asincronia del protocollo MQTT.

Prima di descrivere l'architettura, è bene far notare che i diagrammi UML relativi a ogni componente sono presenti nel file "UML.pdf" allegato a questa relazione, ognuno dei quali è indicato con una didascalia.

L'architettura del software è descritta dal seguente diagramma UML:

Visual Paradigm Standard(20019238_suzzi(Università del Piemonte Orientale))

Figura 0. Diagramma delle classi del sistema



Grazie alla modularità del sistema, è possibile dividere la classe Main in componenti indipendenti e separate. Si è scelto quindi di utilizzare un'architettura multiprocesso, separando il sistema in componenti indipendenti una dall'altra: Database, Controllo, SmartBench, HttpServer e Sensori. Ogni componente funziona tramite l'uso delle classi al suo interno, che descriveremo singolarmente più avanti.

Funzionamento ad alto livello del sistema:

Nel diagramma delle sequenze nel file "UML.pdf" viene rappresentato il funzionamento "tipico" della SmartBench.

Come prima cosa, il tecnico che si occupa di installare la SmartBench accede alla pagina di configurazione posta su un HTTP server. Da questa pagina il tecnico può settare alcuni valori per il funzionamento del sistema. In particolare, questi valori vengono utilizzati dalla componente di Controllo per gestire il periodo di tempo di un allenamento, pausa e monitoraggio del bilanciamento. Fatto questo, la SmartBench è pronta per essere utilizzata. L'utente che vuole accedere alla SmartBench effettua il login da un'interfaccia utente. Dopodiché, il client può comunicare tramite il broker in cloud con le componenti Database e Controllo, in base alla funzione richiesta (inizio allenamento, richiesta dello storico degli allenamenti passati al DB, ecc...). La componente di Controllo comunica strettamente con la componente Sensori e ne gestisce l'attivazione e la disattivazione. La componente Database invece elabora le query in arrivo dal client, e gli restituisce i risultati pubblicandoli sul broker esterno.

Dopo aver descritto il funzionamento del sistema ad alto livello, si può passare ora a descrivere ogni classe all'interno delle componenti.

Descrizione delle classi divise per componenti:**A. Componente Httpserver:**

(Vedere "figura 1" nel file "UML.pdf")

La componente Httpserver contiene le seguenti classi:

- **Main:** la classe main contiene il metodo main() per attivare l'intero sistema. Questo metodo richiama altre due classi Main contenenti i processi per inizializzare le altre componenti del programma: MainC per Controllo, e MainS per Sensori. La classe Main inizializza gli indirizzi IP dei broker in cloud e locale, e tutti i topic che verranno utilizzati dalle classi iniziate dal proprio main, leggendoli da un file in formato JSON. Per quanto riguarda l'attivazione della componente Database, questa viene attivata con un main separato dal resto del sistema, in modo da poterla utilizzare su una Beaglebone diversa, utilizzando il protocollo di cifratura SSL.

Il Main principale si incarica di attivare l'HTTP server sulla porta 11000, richiamare tutte le classi per gestire le richieste HTTP, e di inizializzare:

- i tempi di pubblicazione dei messaggi dai sensori;
- il peso di default impostato sui pesi del bilanciamento;
- la probabilità di invio da parte del bilanciamento del messaggio del suo riposizionamento in sede (usato per scopi di simulazione, non avendo una vera e propria panca piana).

Una parte importante del Main è la lista_thread: questa è appunto una lista contenente tutti i thread dell'applicativo, che possono essere quindi inizializzati, fermati o resettati tramite la pagina di configurazione descritta più avanti. Oltre alla lista_thread abbiamo un thread main_runnable, che si occupa di inizializzare il Controllo della SmartBench e attivare il sistema.

- **RootHandler:** Questa classe mostra a schermo la pagina HTML "conf.html", un'interfaccia da cui è possibile configurare i tempi di pausa, di una singola serie e del monitoraggio del bilanciere, tramite un form. Quando si inviano i dati, tramite un'istruzione di post vengono reindirizzati i dati alla classe EchoPostHandler che li elabora e li passa successivamente all'applicazione.
- **EchoPostHandler:** Questa classe si occupa di gestire i dati inseriti dall'utente per configurare i tempi di esecuzione di ogni serie, il tempo di pausa tra una serie e l'altra e il tempo di monitoring del bilanciere, dopodiché si occupa di settare le variabili della classe Main time_all, time_pause e time_bil con i parametri letti. Nel caso l'utente abbia attivato lo shutdown, questi effettuerà una POST dove il body del messaggio conterrà un comando del tipo "{cmd : error}". L'handler dunque riceverà questo comando, lo parsificherà e setterà la variabile del Main "ERROR" a 1, la quale intima al controllo di fermare tutti i thread e spegnere l'applicazione.
- **ResetHandler:** Questa classe si occupa di gestire il caso in cui l'utente clicchi sul bottone di reset. In questo caso, si attiva il ResetHandler che interrompe tutti i thread presenti nella lista_thread del Main, e setta le variabili del Main (broker, time_all, time_pause e time_bil) a dei valori di default. Dopodiché, il ResetHandler fa ripartire il main_task con questi valori di default che a sua volta, farà partire tutte le thread principali del sistema.

B. Componente Sensori:

(Vedere "Figura 2" nel file UML.pdf)

La componente Sensori contiene le seguenti classi:

- **MainS:** Questo main è un processo che si occupa di inizializzare le classi della componente Sensori: Bracciale, Bilanciere e Pesi.
- **Sensori:** questa non è una classe, ma un raggruppamento di tre classi che hanno lo stesso funzionamento e seguono lo stesso schema: Bracciale, Bilanciere e Pesi. Queste classi sono sottoscritte alla classe Controllo, e hanno ognuna di esse una Callback per agire in base ai comandi che il Controllo le invia. Inoltre ognuna di esse possiede una classe Pub (Bracciale Pub, BilancierePub e PesiPub) che si occupa di leggere i dati simulati da un file e di pubblicare in maniera asincrona e casuale uno tra questi dati, che verranno letti e interpretati dal controllo. I tre sensori sono situati su quattro piedini della Beaglebone diversi: OUT0 per i messaggi di frequenza cardiaca del bracciale, OUT1 per i messaggi di pressione sanguigna del bracciale, OUT2 per i messaggi di alzata e riposizionamento in sede del bilanciere e OUT3 per i messaggi di notifica del peso attuale rilevato dai pesi. Per quanto riguarda il sensore dei Pesi, esso non pubblica dopo un certo intervallo di tempo, ma rimane in attesa di un messaggio di richiesta e modifica del peso corrente da parte del controllo. Questa funzione permette di facilitare la verifica che il peso sia coerente tra le due componenti.

C. Componente Controllo:

La componente Controllo contiene le seguenti classi:

- **MainC:** la classe main del controllo. Tramite il metodo main(), questa classe inizializza le classi della componente Controllo, che lavoreranno in stretto contatto con quelle della componente Sensori.
- **Controllo:** La classe cuore del sistema. Il Controllo si occupa di gestire ed effettuare il vero e proprio allenamento, elaborando tutte le informazioni in arrivo dai sensori e inviandogli dei comandi. Inoltre, si occupa anche di scrivere l'allenamento appena concluso sul Database esterno, pubblicandolo sul broker in cloud. Il Controllo riceve i comandi di start e stop allenamento dal client, oltre ai dati dell'atleta al momento del login.
- Il Controllo gestisce anche la parte di monitoraggio del livello di altezza del bilanciere utilizzando un metodo della classe Monitoring (descritta più avanti): se entro un lasso di tempo time_bil (specificato dal tecnico in fase di configurazione) non vengono rilevate nuove ripetizioni, allora viene interpretata come una situazione di allarme, in quanto l'atleta non è riuscito a sollevare il bilanciere e sta dunque soffocando sotto il suo peso. Il Controllo pubblicherà un messaggio di allarme al client, oltre a fermare tutti i sensori e tutti gli altri thread.
- Il controllo si serve di alcune sottoclassi che generano thread secondarie per gestire task differenti:
- **DBWriter:** Questa classe si occupa di inviare i dati dell'allenamento appena concluso al Database degli allenamenti in modo automatico. Viene richiamata dalla classe Controllo quando si deve scrivere il resoconto dell'allenamento appena concluso.
- **PubToClient:** Questa classe è il "ponte" tra client e Controllo. Si incarica di pubblicare al client i vari messaggi da parte del controllo e delle classi che gestisce. In base al campo "value" al suo interno, PubToClient invia diversi messaggi al client. La PubToClient invia anche altri tipi di messaggi, tra cui quello di "errore" quando non vengono trovati i dati nel Database da passare al client, "logout" quando si effettua il logout, "allarme" per mostrare l'allarme al client, e così via.
- **SmartBench:** Questa classe rappresenta la SmartBench vera e propria, e contiene i metodi per ridurre e settare il peso, e per inviare l'allarme al client. In una (possibile) implementazione reale, si dovrebbe attivare un vero e proprio allarme sonoro per attirare l'attenzione e risolvere l'emergenza, ma questo aspetto non è stato approfondito in quanto si tratta di una simulazione, non avendo a disposizione un macchinario vero e proprio. La notifica di variazione del peso corrente o dell'allarme viene gestita dalla classe PubToClient.
- **Monitoring:** Questa classe ha la funzione di monitorare il corretto svolgimento dell'allenamento. In particolare, un contatore viene incrementato ogni secondo che passa tra l'arrivo di una ripetizione e un'altra. Quando un nuovo messaggio di ripetizione avvenuta arriva al Controllo, il contatore viene resettato a zero. In questo modo si ha un allarme solo quando non arriva nessun messaggio per time_bil secondi (valore che, se superato dal contatore del Monitoring, indica una situazione di emergenza. Se non viene raggiunto il limite di tempo di monitoring, allora si è in una situazione di normale svolgimento

dell'allenamento. Durante la pausa tra una serie e un'altra, in modo da non avere falsi allarmi, il contatore viene bloccato e resettato a zero.

- **ControlloLogger:** Questa classe consiste in un semplice logger che registra su un file "logfile.log" i comandi di sistema, la pubblicazione e la ricezione dei messaggi MQTT della classe Controllo e gli eventi scaturiti da quest'ultimi.

D. Componente Database:

(Vedere "Figura 4" del file UML.pdf)

La componente Database contiene le seguenti classi:

- **MainDB:** questa classe main, come MainC e MainS, si occupa di inizializzare ed eseguire il codice delle varie classi appartenente alla componente Database.
- **Database:** Come per i sensori, ci sono tre diversi tipi di classi database:
 - a. **DB_Utenti:** questo database contiene i metodi per elaborare e gestire i dati di login degli utenti. Quando un utente vuole effettuare un login, il client manda la richiesta al Database, che confronta il nome utente con quelli all'interno del file "utenti.xml". Se trova una corrispondenza, restituisce al client e al controllo (per fargli inizializzare tutte le sottoclassi che gestisce) un ACK di conferma che l'Account con cui ci si è loggati nel client è all'interno del database, altrimenti ritorna un messaggio di errore, che viene mostrato sull'interfaccia grafica del client nella pagina index.html.
 - b. **DB_Allenamenti:** questo DB contiene tutti i metodi per cercare e inserire i resoconti dei vari allenamenti conclusi. Il Controllo, una volta terminato l'allenamento, lo inoltra al DB che con l'opportuno metodo lo scrive nel file "allenamenti.xml". Allo stesso modo, il DB può ricercare all'interno dello stesso file tutti gli allenamenti eseguiti da utente, oppure solamente il più recente, in base a ciò che il client aveva richiesto.
 - c. **DB_Piani:** questo DB contiene i metodi per ricercare, pubblicare e verificare i piani di allenamento di ogni atleta. Solo un trainer può aggiungere un nuovo piano a questo database, mentre un atleta può solo visualizzare il piano attivo che deve seguire. Quando si inserisce un nuovo piano, vengono invalidati quelli precedenti, in modo che non ci sia più di un solo piano valido per utente.
- **GestoreDB:** Questa classe si occupa di richiamare i metodi dei vari Database per parsificare ed eseguire i comandi in arrivo dal client. Il Gestore possiede una sua classe Callback per ricevere i messaggi dal Controllo e dal client, e una classe Pub per inviare i valori di ritorno calcolati dai metodi del DB quando viene richiamato.
- **Account:** una delle tre classi base, l'Account indica i parametri per riconoscere un Utente al momento del login, e contiene i seguenti campi:
 - *username*: il nome utente dell'utente.
 - *password*: la password dell'utente.
 - *tipo*: indica il tipo di utente. Può essere Atleta o Trainer.

(Vedere "Figura 3" del file UML.pdf)

E. Componente SmartBench:

(Vedere “Figura 5” del file UML.pdf)

La componente SmartBench non è una vera e propria componente, in quanto non possiede un Main che ne inizializza i processi, ma è separata dalle altre componenti in quanto le altre componenti dipendono dalle classi contenute in essa:

- **Allenamento:** Una classe “base”, ovvero senza metodi che influiscono sul funzionamento del sistema. Questa classe contiene tutti i campi che può avere un allenamento:
 - *nomeAtleta*: lo username dell’atleta che ha eseguito questo allenamento;
 - *valutazione*: calcolata dal controllo al termine dell’allenamento, si basa su quante ripetizioni ha effettuato l’atleta. Se esegue tutte le ripetizioni provviste dal piano senza riduzione di peso, allora prende il voto massimo. Il voto è in una scala da 1 a 10.
 - *mediaFC*: rappresenta la media della frequenza cardiaca dell’atleta rilevata durante l’allenamento.
 - *mediaPS*: rappresenta la media della pressione cardiaca dell’atleta rilevata durante l’allenamento.
 - *mediaRip*: rappresenta la media delle ripetizioni effettuate per ogni serie dall’atleta durante l’allenamento.
 - *mediaPeso*: rappresenta la media del peso utilizzato dall’atleta durante l’allenamento. Utile al trainer per “correggere il tiro” riguardo al peso sollevabile dall’atleta: se la media è di molto inferiore al peso stabilito dal piano, allora il trainer dovrà modificare il piano di conseguenza.
 - *ripetizioniAll*: Indica il numero di ripetizioni totali eseguite durante l’allenamento.
 - *pesoMaxAll*: indica il peso iniziale stabilito dal piano di allenamento.
 - *timestamp*: indica il momento in cui l’atleta ha completato l’allenamento.
- **Piano:** un’altra classe “base”, il Piano rappresenta un piano di allenamento scritto dal Trainer che l’utente deve seguire. Contiene i seguenti campi :
 - *nomeUtente*: indica lo username dell’atleta a cui il piano è associato.
 - *validità*: indica se il piano è ancora valido (valid) o meno (invalid).
 - *totRip*: indica il numero di ripetizioni da eseguire per ogni serie.
 - *peso*: indica il peso iniziale dell’allenamento.
 - *serie*: indica il numero di serie da eseguire durante l’allenamento.

NB: Come scritto all’inizio di questo capitolo, per poter descrivere con chiarezza l’implementazione svolta e le logiche scelte per il sistema è inoltre stato scelto di utilizzare un approccio UML, andando a integrare le informazioni presenti nel paragrafo precedente. I diagrammi sono reperibili nel file “UML.pdf” allegato.

Descrizione dell'Interfaccia Utente

Interfaccia di Configurazione

L'interfaccia di configurazione viene visualizzata dall'HTTP Server al momento dell'avvio del sistema, e permette di inizializzare l'applicazione.

Questa interfaccia è pensata per essere acceduta solo da tecnici esperti durante la configurazione del sistema. Permette di configurare l'intervallo di tempo di esecuzione di una serie dell'allenamento, quello di pausa tra una serie e l'altra, e quello di monitoraggio del livello del bilanciere. Inoltre, sono presenti due bottoni di "reset" e "shutdown", le cui funzioni sono state descritte precedentemente.

In caso un utente avesse eseguito il login, con il comando di reset, invio e shutdown questi verranno reindirizzati alla pagina di login, poichè si è ipotizzato che queste operazioni debbano essere fatte in condizioni di pericolo oppure in caso vi siano malfunzionamenti della macchina.

CONFIGURAZIONE SMARTBENCH

Wellness first

IP Broker

193.206.55.23

Tempo di esecuzione di ogni serie

60s

Tempo di pausa tra ogni serie

60s

Tempo di monitoraggio dello stato del bilanciere

10s

Invio

Per resettare l'applicazione, premere su "Reset"

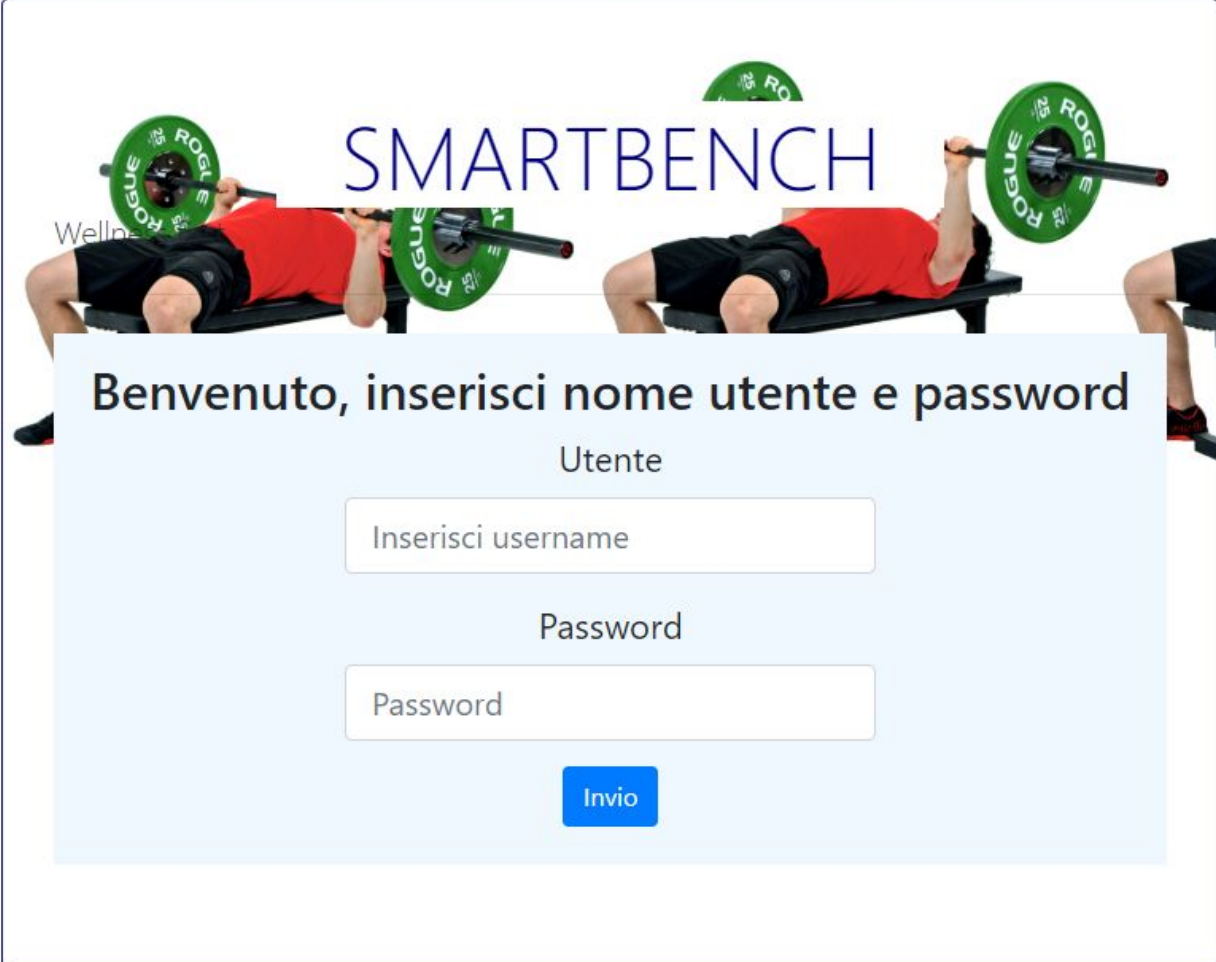
RESET

Interruttore di emergenza

SHUTDOWN

I valori all'interno dei campi sono quelli di default.

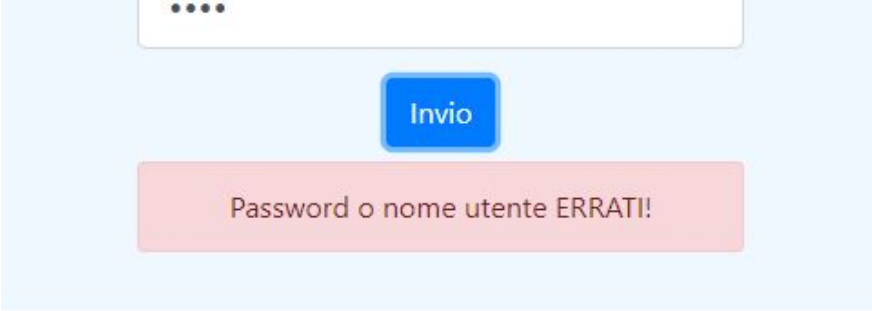
Pagina di login



The image shows the login page for 'SMARTBENCH'. The background features a person in a red shirt and black shorts performing a bench press with a barbell loaded with green 'ROGUE' weights. The word 'SMARTBENCH' is displayed in large, blue, sans-serif capital letters across the top. Below the title, the text 'Benvenuto, inserisci nome utente e password' (Welcome, enter username and password) is centered. Underneath, there are two input fields: the first is labeled 'Utente' (User) and contains the placeholder text 'Inserisci username'; the second is labeled 'Password' and contains the placeholder text 'Password'. A blue button labeled 'Invio' (Submit) is positioned below the password field. The entire login form is overlaid on a light blue rectangular background.

Permette all'utente di effettuare il login alla SmartBench inserendo i propri dati di username e password.

Se si inseriscono dei dati non validi viene ritornato un NACK dal database e appare un alert in prima pagina che obbliga ad inserire dei dati corretti.



This image shows a close-up of the login form after an incorrect login attempt. The password input field is now filled with four black dots. Below the 'Invio' button, a red rectangular alert box contains the text 'Password o nome utente ERRATI!' (Password or username ERRATI!).

Se i dati inseriti, invece, sono corretti allora si viene reindirizzati alla pagina loginAtleta.html se il proprio utente è di tipo atleta, altrimenti si viene reindirizzati alla pagina loginTrainer.html.

Interfaccia Grafica per l'utente

Benvenuto iera97!

Cosa vuoi fare atleta?

Inizia allenamento

Storico allenamenti

Piano di allenamento attuale

Performance ultimo allenamento

Effettua il logout

Piano di allenamento

Nome: iera97


Validita': valid

Ripetizioni da fare: 30

Peso consigliato: 20

Numero di serie totali: 2

Stato Bilanciere:



Bilanciere in sede..

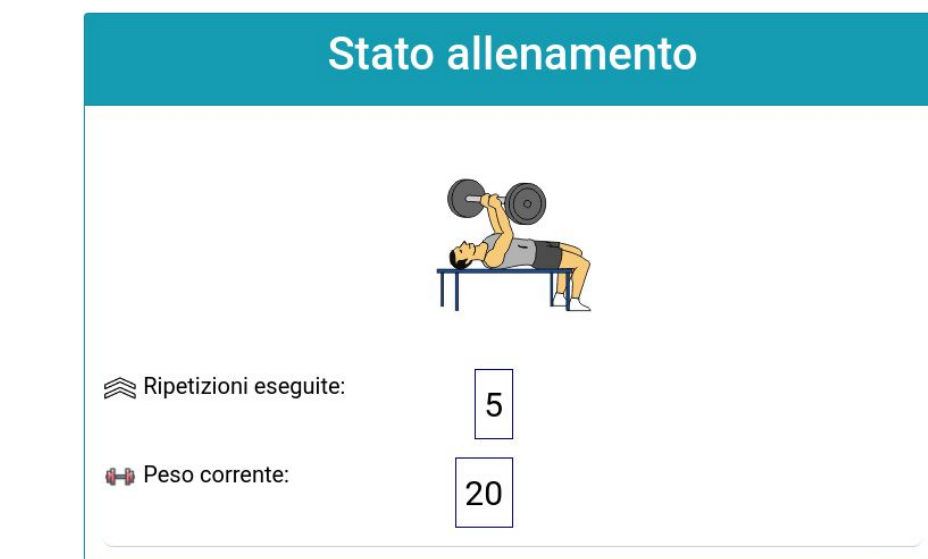
L'interfaccia grafica permette all'utente di effettuare, in base al tipo a cui appartiene (Trainer o Atleta), azioni diverse sulla SmartBench e sul Database. Può essere acceduta in locale tramite il browser e funziona sfruttando le librerie PAHO per JavaScript. L'interfaccia grafica utente appena carica richiede la scheda contenente il piano dell'utente che ha appena eseguito il login e la visualizza in un box a lato della pagina.

L'atleta può scegliere tra varie opzioni di:

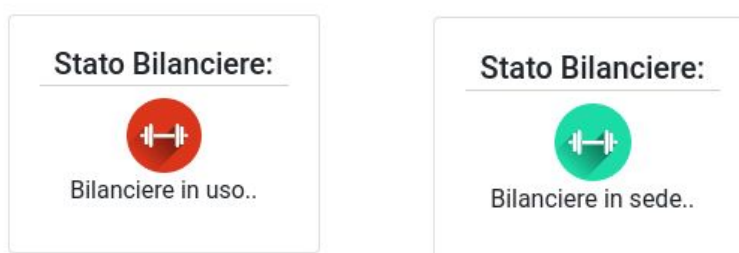
- *Iniziare l'allenamento*

Alla pressione di questo pulsante ha inizio l'allenamento. Si è scelto di non rendere possibile visualizzare lo storico o azioni relative al database durante l'allenamento, se non quella di interrompere l'allenamento attuale. Questo perché un atleta durante l'allenamento non ha bisogno di visualizzare i propri risultati passati.

Durante l'allenamento, però, è possibile visualizzare le proprie performance in tempo reale: l'immagine animata dell'atleta che esegue le alzate del bilanciere in tempo reale, il numero di ripetizioni eseguite che viene incrementato ogni volta che l'atleta esegue una ripetizione e il peso corrente, che può diminuire in caso l'atleta stia avendo delle performance non adeguate.




Durante l'allenamento l'atleta può soltanto interrompere l'esecuzione oppure riposizionare il bilanciere al proprio posto per terminare la serie e iniziare la pausa che sarà determinata dal tecnico in fase di configurazione. Ciò viene mostrato con un'icona con sfondo verde se il bilanciere è riposto correttamente nella propria sede, oppure con sfondo rosso se il bilanciere è in uso.



Una volta terminato l'allenamento verrà mostrato all'atleta il resoconto dell'allenamento appena concluso.

Stato allenamento



 Ripetizioni eseguite:

26

 Peso corrente:

15

Nome: iera97

Valutazione: 4

Frequenza_cardiaca_media: 78.0

Pressione_Sanguigna_media: 110

Numero_di_ripetizioni_medie: 13

Peso_medio: 17

Ripetizioni_allenamento: 26

Peso_massimo_allenamento: 20

Timestamp: 2019/07/09,13:05

FINE allenamento

Sia a fine allenamento che ad allenamento interrotto è di nuovo possibile selezionare le altre opzioni, tra cui le statistiche della performance di un allenamento appena eseguito, dato che questo viene registrato automaticamente sul database in cloud.

Un'altra possibilità è il monitoring del bilanciere. Se non viene eseguita un'alzata del bilanciere prima di un certo tempo di monitoring (definito in configurazione dal tecnico) allora verrà visualizzato un allarme in interfaccia con un relativo messaggio di chiamata dei soccorsi. Si potrebbe applicare lo stesso a livello hardware con un segnale sonoro ma per problemi di costi, ci si è limitati alla sola visualizzazione a livello client.

Stato allenamento



Allenamento terminato, chiamare subito i soccorsi

- Visualizzare lo storico degli allenamenti
- Visualizzare le performance dell'ultimo allenamento eseguito
- Visualizzare il piano attuale

Storico allenamenti	
Nome: iera97	
Valutazione: 10	
Frequenza_cardiaca_media: 76.46875	
Pressione_Sanguigna_media: 109	
Numero_di_ripetizioni_medie: 46	
Peso_medio: 20	
Ripetizioni_allenamento: 92	
Peso_massimo_allenamento: 20	
Timestamp: 2019/07/08,14:51	
Nome: iera97	
Valutazione: 7	
Frequenza_cardiaca_media: 79.53333333333333	
Pressione_Sanguigna_media: 109	
Numero_di_ripetizioni_medie: 22	
Peso_medio: 17	
Ripetizioni_allenamento: 44	
Peso_massimo_allenamento: 20	
Timestamp: 2019/07/08,14:47	

Immagine rappresentativa degli ultimi tre punti

- Effettuare il logout

Quando l'utente decide di eseguire il logout, viene mostrato a schermo un prompt di conferma per evitare click accidentali. Cliccando su "sì", l'utente verrà reindirizzato alla pagina di login e dovrà rifeffettuare l'accesso, altrimenti rimarrà sulla pagina corrente.



La pagina principale del trainer invece è strutturata in modo simile a quella dell'atleta, ma presenta funzionalità maggiori.

Benvenuto marco96!

Cosa vuoi fare, trainer?

Inizia allenamento

Vedi Piano di allenamento attuale

Inserisci un nuovo piano di allenamento

Storico Piani

Effettua il logout

Piano di allenamento

Nome: marco96


Validita': valid

Ripetizioni da fare: 30

Peso consigliato: 20

Numero di serie totali: 3

Stato Bilanciere:



Bilanciere in sede..

Il trainer può scegliere di:

- *Iniziare l'allenamento*, seguendo lo stesso modello di quello descritto per l'atleta.
- *Visualizzare l'ultimo piano di allenamento valido per un determinato utente*
- *Visualizzare lo storico di tutti i piani di un certo utente* (compresi quelli non piu' validi)

Inserisci l'utente di cui vuoi
cercare l'ultimo piano

Visualizza ultimo piano

Nome: iera97
Validita': valid
Ripetizioni da fare: 30
Peso consigliato: 20
Numero di serie totali: 2

Se l'utente ricerca non viene trovato, viene ritornato un messaggio di default "Nessuno presente".

prova

Invia

Visualizza ultimo piano

Nessuno presente

- Inserire un nuovo piano di allenamento di un utente.

Inserisci il nuovo piano di allenamento

Username

Inserisci username

Ripetizioni totali per serie

Inserisci numero di ripetizioni da fare per ogni s

Peso

Inserisci peso

Numero di serie da eseguire

Inserisci numero di serie da effettuare

Invio

Quando il trainer sceglie questa opzione, viene mostrato un form contenente dei campi dove devono essere inseriti i dati relativi al piano che si vuole aggiungere. Nel caso uno o più campi risultino vuoti al momento dell'invio, viene mostrato un alert.

Inserisci numero di serie da effettuare

Invio

Errore: Campi vuoti!

Inoltre viene mostrato un alert differente se il formato dei dati non è corretto.

ciao

Invio

Errore: Inserimento dei dati nei campi errato

- Effettuare il logout, seguendo lo stesso schema presente per l'atleta.

Validazione del software strutturato

Non essendo possibile una validazione vera e propria del software, in quanto ottenere una panca piana sarebbe risultato eccessivamente costoso, i valori rilevati dai sensori sono semplicemente letti da dei file .xml o impostati leggendo un file .json.

Per la simulazione del riposizionamento in sede del bilanciante, è stato definito un algoritmo che, dalla durata di una serie e dal numero di ripetizioni effettuate fino a quell'istante, definisce una distribuzione di probabilità. Questa indica la probabilità che l'atleta riposizioni il bilanciante prima dello scadere del tempo di allenamento. All'aumentare delle ripetizioni, cresce anche la probabilità di riposizionamento, quindi si è pensato che l'atleta aumenti il proprio livello di fatica ad ogni ripetizione. Questo si traduce nell'algoritmo in un rapporto tra il tempo di allenamento e il numero di ripetizioni, moltiplicato per un valore di probabilità letto da un file .json. Questo valore garantisce di poter gestire l'eventuale aumento o diminuzione di probabilità di riporre il bilanciante semplicemente cambiandolo nel file.

Per la simulazione del peso, si passa un peso di default tramite file .json e successivamente si confronta nella classe di controllo il valore registrato col valore effettivo, per garantire che non vi siano incongruenze di peso.

Per la simulazione del sensore bracciale sono stati definiti dei valori in un file .xml con un valore medio di 110 e una varianza di 30 per la pressione, in modo da generare valori piuttosto eterogenei. Inoltre, per evitare che vengano letti valori iterativamente e quindi generare valori uguali ad ogni resoconto, è stato introdotto un fattore probabilistico che va a prelevare i valori in ordine casuale tra i 1000 registrati nel file.

Per i test relativi alla parte di allarmistica è stata aggiunta la possibilità, in fase di configurazione, di definire un monitoring che generi subito l'allarme e che potrebbe essere utile in fase di testing reale dell'apparecchio hardware. Inoltre, per la simulazione software è stato impostato un tempo di invio di una ripetizione, letto da file .json, maggiore rispetto a quello di monitoring in modo tale che questo possa inviare un segnale d'allarme il prima possibile.

Per il testing del client sono stati effettuati ripetutamente start e stop dell'allenamento per verificare l'effettiva capacità e prontezza del software ad un utente poco attento. Non sono infatti emersi problemi dovuti a questa esagerata richiesta dal client.

Un altro controllo relativo al client è la possibilità che in esecuzione dell'allenamento alcuni sensori potessero mandare dei valori un momento dopo l'interruzione dell'allenamento, e che quindi generassero incongruenze tra quello che succede nel sistema e quello che effettivamente l'utente vede. Infatti sono stati aggiunti dei controlli soprattutto sul sensore del bilanciante, in modo che non possano essere contate delle ripetizioni avvenute un momento dopo l'interruzione dell'allenamento o la fine della serie per garantire un resoconto di allenamento coerente con quello di una situazione reale. Per il testing del database è stato necessario confrontare i risultati inviati al client dalle classi di DB, con quelli effettivamente registrati nel file .xml.

Bibliografia

- [1] https://it.wikipedia.org/wiki/Representational_State_Transfer
- [2] <https://stackoverflow.com>
- [3] https://www.slideshare.net/omnys_keynotes/mqtt-il-protocollo-che-rende-possibile-l-internet-of-things
- [4] <http://www.html.it/>
- [5] [Slide del corso disponibili sul DIR](#)
- [6] <https://www.w3schools.com>