INTRODUZIONE

L’iterazione 3 è dedicata ai casi d’uso legati all’aggiornamento e visualizzazione dello stato dei macchinari, ovvero UC-10 e UC-11.

ASSUNZIONI FATTE

Per poter conoscere lo stato attuale di un macchinario si è deciso di assumere innanzitutto che su ognuno di essi siano presenti tre pulsanti:

* Libera Macchinario
* Occupa Macchinario
* Macchinario Guasto

Che comunicano al server lo stato corrente quando premuti da un utente. In vista del caso d’uso legato alla generazione di una scheda fitness aggiornata in base allo stato corrente dei macchinari, si è scelto di inviare le informazioni nel formato:

* Nome Macchinario (stringa)
* Tipologia Macchinario (stringa)
* Orario al quale si libera il macchinario (stringa)

Dove l’informazione legata all’orario assumerebbe valore nel seguente modo:

* Premendo “LIBERA MACCHINARIO” viene inviato al server l’orario attuale;
* Premendo “OCCUPA MACCHINARIO” viene calcolato internamente al macchinario e poi inviato , la cui stima X potrebbe essere fatta osservando l’occupazione media di quel macchinario.
* Premendo “MACCHINARIO GUASTO” viene inviato al server l’orario , assicurando la minore priorità possibile.

In questo modo è possibile gestire un sistema di priorità tra i macchinari utilizzando l’orario dal quale sono/saranno liberi.

IMPLEMENTAZIONE

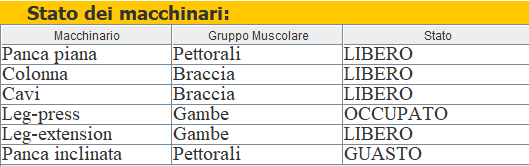
Per permettere agli utenti e trainer di visualizzare lo stato corrente di macchinari è stato creato un client “Dashboard” che si occupa di mostrare a schermo in formato tabellare le informazioni sui macchinari, aggiornandole ogni 5 secondi.

Implementando la Dashboard si è usato il paradigma MVC creando una View e un Controller che, all’avvio, invia periodicamente una richiesta all’URL:

http://localhost:8080/getStatoMacchinari

processando la stringa JSON ricevuta in risposta attraverso il metodo di parsing “parsingMacchinari(String s)” e inviando le informazioni alla View. La classe “Macchinario” di questo client è stata modificata in modo tale da avere il metodo “getStato()” in grado di restituire la stringa “OCCUPATO”, “LIBERO” o “GUASTO” confrontando il timestamp del macchinario con l’orario attuale.

Il risultato della View per il client Dashboard è quindi, ad esempio:



Server-side invece è stato creato il componente “*Controller Aggregatore”* come REST Controller. Esso contiene un ArrayList di macchinari che aggiorna ogni volta che riceve nuove informazioni da un macchinario.

Questo Controller mette a disposizione due API, una per permettere alla Dashboard di leggere la lista di macchinari aggiornata allo stato corrente ed una per permettere ad ogni macchinario di aggiornare il proprio stato memorizzato sulla lista:

L’API “InfoMachineryIF” che contiene il metodo

@GetMapping(“/getStatoMacchinari”)

RisorsaJSON getStatoMacchinari()

L’API “StatusIF” che contiene il metodo

@GetMapping(“/writeStatoMacchinario”)

writeStatoMacchinario()

CASI DI TEST

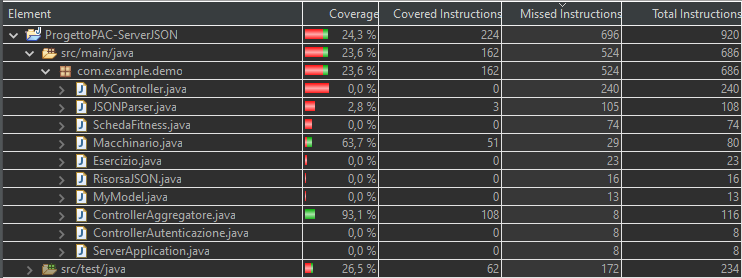
I metodi che richiedono unit-testing sono: lato server il metodo “writeStatoMacchinario()” in quanto “getStatoMacchinari()” si limita a ritornare una lista già esistente; il lato client (Dashboard) ha invece solo il metodo “parsingMacchiari()” che si occupa di ritornare una lista di macchinari partendo dalla stringa JSON ricevuta dal Controller Aggregatore.

Per testare il metodo “writeStatoMacchinario()” si è dato in input diversi macchinari, eseguendo quindi il metodo più volte e controllando poi che la lista venga correttamente aggiornata.

Per testare il metodo “parsingMacchinari()” della Dashboard si è dato in input una stringa JSON rappresentante diversi macchinari controllando che la lista ritornata fosse corretta.

COPERTURA DEI CASI DI TEST

Test del Controller Aggregatore:



Test del Controller Dashboard:

