INTRODUZIONE

L’iterazione 5 è stata dedicata allo sviluppo e all’implementazione dei casi d’uso relativi alla richiesta di aggiornamento della scheda da parte dell’utente sulla base dello stato attuale dei macchinari. I casi d’uso in questione sono l’8 e il 9.

IMPLEMENTAZIONE

Dal punto di vista dell’app dell’utente è stata aggiornata la View aggiungendo i pulsanti “Scheda Originale” e “Scheda Aggiornata” per permettere all’utente di passare tra una versione e l’altra della propria scheda fitness. I campi testuali dove vengono visualizzati gli esercizi della scheda aggiornata sono gli stessi di quelli usati per la scheda originale ed è per questo che non sono richieste ulteriori modifiche alla View.

Sul Controller del client sono stati aggiunti i rispettivi *Action Listener* per inviare le richieste tramite URL al componente lato server denominato *Manager Schede* (REST Controller).

Il *Manager Schede* è già stato implementato nell’iterazione precedente per supportare i casi d’uso relativi alla richiesta di schede fitness originali da ritornare all’utente, perciò, nel contesto dell’iterazione, è sufficiente aggiungere una API REST per rispondere alle richieste di aggiornamento delle schede:

@GetMapping(“/aggiornaScheda”)

public RisorsaJSON aggiornaScheda(int id)

Il metodo aggiornaScheda(), ricevendo l’id dell’utente, richiede sia la rispettiva scheda fitness al DB sia la lista dei macchinari con lo stato attuale per produrre la nuova lista di esercizi.

RisorsaJSON è un’istanza contenente sia l’ID della richiesta sia la stringa JSON rappresentante la lista di esercizi.

Esempio di RisorsaJSON (in formato JSON) inviata con id=1 e un solo esercizio nella lista:

{"lista":"[{\"tipo\":\"Braccia\",\"nome\":\"Flessioni\"}]”, “id”:1}

Essendo il “fetching” della scheda fitness dal database un’operazione onerosa (considerato che la richiesta di aggiornamento scheda può essere effettuata più volte in qualsiasi momento) è stata creata una lista dinamica di SchedaFitness all’interno del controller Manager Schede. In questo modo una scheda fitness viene estratta dal DB solo la prima volta per poi essere salvata in questa lista permettendo un accesso molto più veloce nel caso debba essere aggiornata in seguito.

La lista di macchinari con lo stato aggiornato viene ottenuta tramite il *Controller Aggregatore* usando la stessa API usata dalla Dashboard: *InfoMachineryIF*.

IMPLEMENTAZIONE ALGORITMO

Una volta che la richiesta viene ricevuta dal metodo aggiornaScheda() l’effettiva creazione della lista di esercizi viene affidata ad un metodo implementato secondo la metodologia *Greedy*:

algoritmoGreedy(SchedaFitness S, List<Macchinario> Lm)

L’output è una lista di macchinari che viene convertita in esercizi per l’invio al client dell’utente.

PSEUDOCODICE ALGORITMO

algoritmoGreedy(EserciziScheda S [1..n], Lista macchinari [1..m])

Lista aggiornata // Contiene i macchinari selezionati dall’algoritmo

Coda1…K // È creata una PriorityQueue per ogni categoria “k” di macchinari

foreach(macchinario m[k]) // AMBIGUO

C[k].enqueue(m);

foreach(Esercizio E[k] in S)

if (M[k] libero) {

aggiornata.add(M)

// M[k] è il macchinario corrispondente a quell’esercizio, se questo è libero

// non verrà sostituito con un altro a prescindere dalla priorità

} else {

Macchinario M <-- Coda[k].dequeue();

aggiornata.add(M);

}

return aggiornata;

COMPLESSITÀ ALGORITMO

Usando la PriorityQueue del JCF le operazioni di enqueue() e dequeue() risultano avere costo , dove in questo caso “m” è il numero di macchinari contenuti in esse; usando invece una LinkedList per contenere la lista risultante “aggiornata” permette di avere costo unitario per l’operazione di inserimento “add()”.

La prima sezione dell’algoritmo, con la creazione e riempimento delle code, ha quindi costo immaginando come caso peggiore quello per cui tutte le code devono essere riempite con tutti i macchinari.

La seconda sezione, invece, partendo con l’iterazione sugli “n” esercizi della scheda utente, esegue “n” volte l’operazione di “dequeue()” e di “add()” per un costo totale di . Questo considerando ancora il caso peggiore, ovvero il caso per cui viene sempre eseguito il ramo “else”.

Complessivamente, il costo dell’algoritmo è: ; siccome si immagina di avere un numero di macchinari “m” maggiore rispetto al numero “n” di esercizi nella scheda, si può approssimare a: .

CASI DI TEST

I nuovi metodi implementati lato server che richiedono unit-testing sono:

* L’algoritmo greedy;
* Il metodo macchinarioLibero() per controllare che un certo macchinario sia libero;
* Il metodo searchScheda() per ritornare una scheda dalla lista dinamica memorizzata sul controller “Manager Schede”.

TEST algoritmoGreedy() & macchinarioLibero()

Dati in input una scheda con determinati esercizi e una lista di macchinari con determinati orari ai quali si libereranno, si verifica che l’algoritmo dia in output una lista di macchinari che:

1. Abbia sostituito correttamente per ogni esercizio un macchinario della stessa categoria, mantenendo l’ordine della scheda originale;
2. Per ogni categoria sia stato assegnato quello a priorità maggiore (ovvero quello con il timestamp minore);
3. Nel caso l’esercizio avesse il macchinario corrispondente libero al momento dell’esecuzione, questo non viene stato sostituito con un altro macchinario equivalente, questo test include quindi quello per “macchinarioLibero()”.

TEST searchScheda()

Sono state aggiunte alla lista di schede del Controller “Manager Schede” diverse schede con i rispettivi ID; dando in input al metodo “searchScheda(int id)” un id in particolare deve essere ritornata in output la scheda corretta quando questa esiste nella lista, “null” quando non esiste.

COPERTURA DEI CASI DI TEST

