Ingegneria del Software A.A. 2016/2017 Esame 2017-07-13

Esercizio 1 (6 punti)

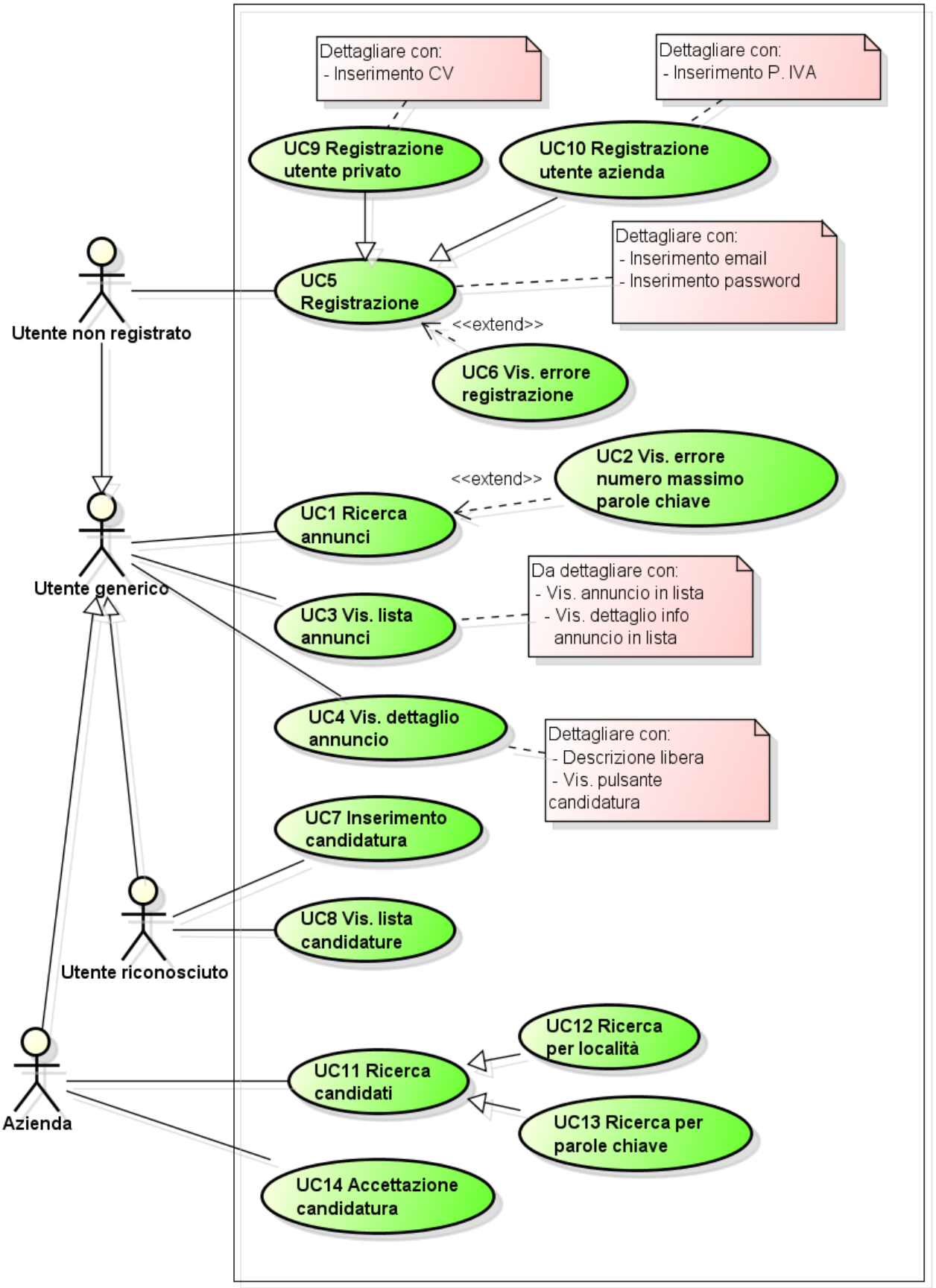
Descrizione

Monster è uno dei principali siti sul mondo del lavoro al mondo. Al suo interno chiunque può ricercare tra le offerte di lavoro esistenti. Le aziende, invece possono inserire le posizioni aperte che stanno ricercando. Qualsiasi utente può ricercare gli annunci disponibili semplicemente inserirendo da una a cinque parole chiave o una località di riferimento. Il risultati di questa ricerca è una lista di annunci, che per ciascuno di essi riporta un titolo, l'azienda che propone l'annuncio e la posizione geografica del lavoro. Il dettaglio di un movimento visualizza una descrizione libera più lunga e la possibilità di candidarsi. Per potersi candidare un utente deve essere registrato. La registrazione richiede l'inserimento di un'email valida, di una password e di un *curriculum vitae*. La candidatura invia un'email all'utente e la aggiunge alla sua lista delle candidature. Anche un'azienda per poter inserire annunci deve registrarsi, inserendo anche la partita iva. Un'azienda può ricercare gli altri utenti per località o per parole chiave. Infine, può accettare la candidatura di un utente ad un annucio di lavoro. Il sistema invierà un'email per segnalare l'accettazione.

Si utilizzino i diagrammi dei casi d'uso per modellare gli scenari descritti. Non è richiesta la descrizione testuale dei casi d'uso individuati.

Soluzione

Dove è riportato il commento "Da dettagliare" devono essere individuati obbligatoriamente i sottocasi d'uso riportati nel testo.


Dettagliare con:
- Inserimento CV
Dettagliare con:
- Inserimento P. IVA
UC9 Registrazione utente privato
UC10 Registrazione utente azienda
Dettagliare con:
- Inserimento email
- Inserimento password
UC5 Registrazione
Utente non registrato
<<extend>>
UC6 Vis. errore registrazione
<<extend>>
UC2 Vis. errore numero massimo parole chiave
UC1 Ricerca annunci
Utente generico
Da dettagliare con:
- Vis. annuncio in lista
- Vis. dettaglio info annuncio in lista
UC3 Vis. lista annunci
UC4 Vis. dettaglio annuncio
Dettagliare con:
- Descrizione libera
- Vis. pulsante candidatura
UC7 Inserimento candidatura
UC8 Vis. lista candidature
Utente riconosciuto
UC12 Ricerca per località
UC11 Ricerca candidati
UC13 Ricerca per parole chiave
Azienda
UC14 Accettazione candidatura

Esercizio 2 (7 punti)

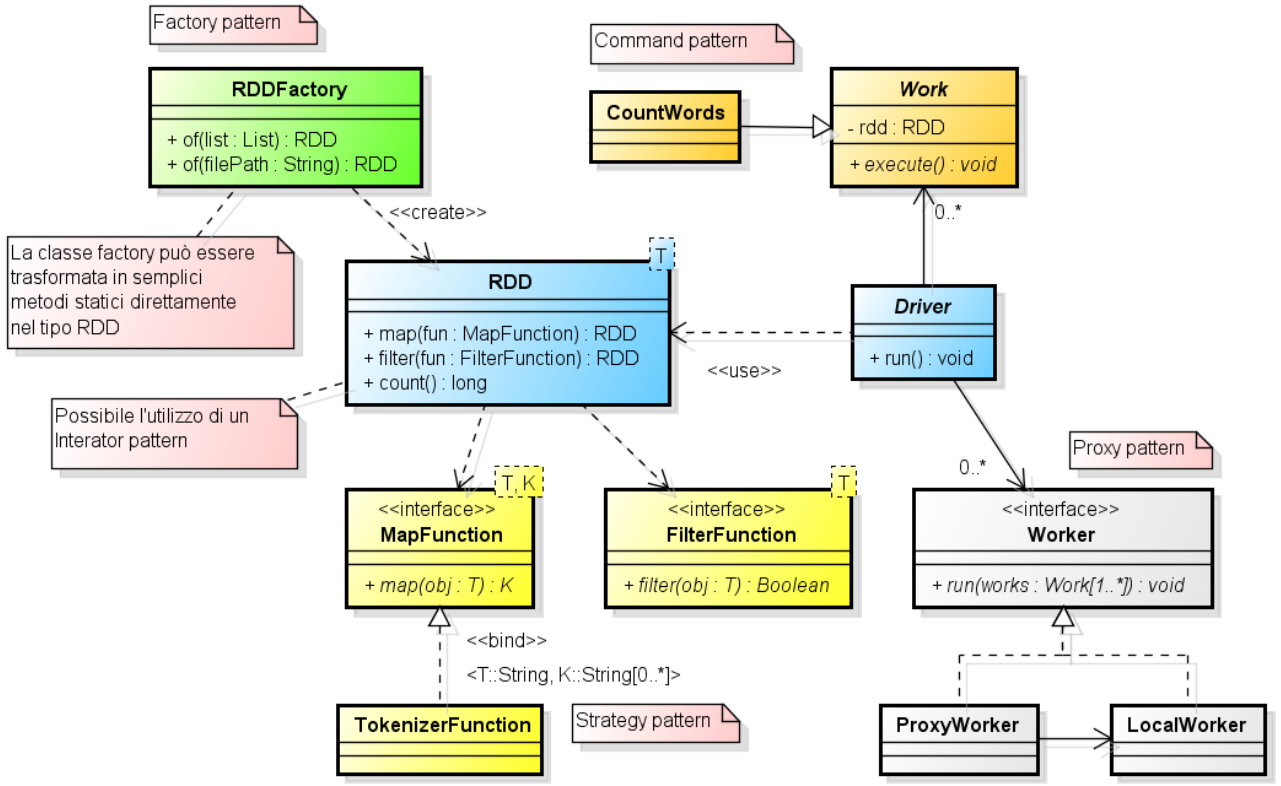
Descrizione

Apache Spark è un motore di esecuzione per processi che elaborano grandi quantità di dati. Il suo punto di forza principale è quello di essere in grado di parallelizzare l'esecuzione di algoritmi in modo distribuito, pur mantenendo un'API semplice e per lo più tipo sequenziale. Il fulcro principale è il tipo `RDD[T]`. Questa struttura può essere pensata tranquillamente come ad una lista di elementi di tipo `T`, ma distribuita fra più macchine, dette *worker*. La lista viene però dichiarata su una macchina principale, detta *driver*, che è anche quella che esegue il programma principale. La costruizione di un RDD avviene utilizzando opportuni metodi, che ne creano una nuova a partire da diverse sorgenti: ad esempio da un oggetto di tipo `List[T]`o da un file creando un `RDD[String]`. La lista espone operazioni come `map` e `filter`. Entrambi questi metodi hanno in input un algoritmo che lavora su un elemento di tipo `T`. La differenza è che `map` trasforma un oggetto di tipo `T` in un oggetto di tipo `K`, mentre `filter` dato un oggetto di tipo `T`, ritorna un valore booleano vero o falso. Il metodo `run` del programma *driver*, quindi, si occupa di inviare le istruzioni da eseguire su una porzione di `RDD` ad ogni *worker*. Poiché il programma da eseguire non ha una struttura fissa, è necessario utilizzare una struttura che ne permetta l'esecuzione in modo uniforme. Si ricorda che l'esecuzione sui *worker* avviene in modo remoto rispetto al *driver*.

Si modelli il sistema descritto utilizzando un diagramma delle classi e gli opportuni *design pattern*. Inoltre, si descriva utilizzando un diagramma di sequenza un programma che conta quante parole sono presenti all'interno di un file di testo. Il file verrà dapprima letto riga per riga, poi suddiviso in parole, che alla fine saranno conteggiate con un opportuno metodo `count`.

Soluzione

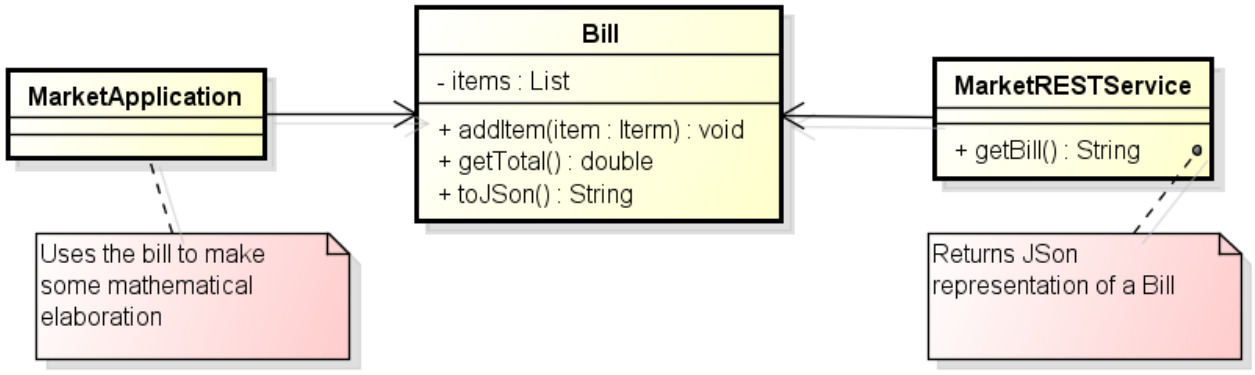
La soluzione prevede l'utilizzo dei seguenti design pattern: Command pattern, Proxy pattern, Strategy pattern, Factory Method pattern.



Esercizio 3 (3 punti)

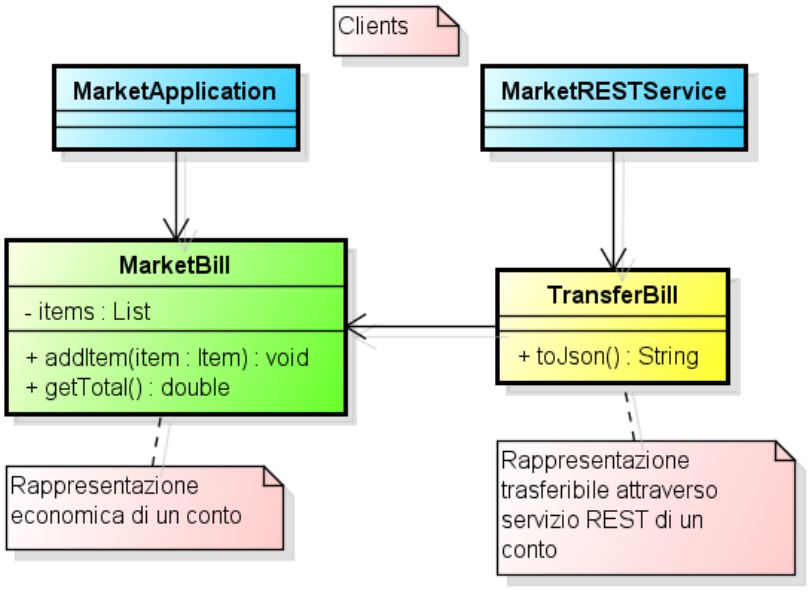
Descrizione

La classe `Bill` è utilizzata per gestire oggetti che rappresentano liste della spesa sia dal punto di vista del calcolo matematico, sia da una applicazione che espone queste informazioni in formato Json tramite un servizio REST. Dato il seguente diagramma delle classi, se ne fornisca una versione modificata in modo tale che la soluzione aderisca al "Single Responsibility Principle".



Soluzione

Le funzionalità del tipo `Bill` devono essere suddivise in opportuni tipi, in modo da suddividere gli assi di cambiamento. In questo modo, se la rappresentazione Json di un conto dovesse cambiare, `MarketBill` non verrebbe modifcato. D'altra parte, se l'algoritmo di calcolo del totale dovesse cambiare, `TransferBill` non verrebbe modificato.


Clients
MarketApplication
MarketRESTService
MarketBill
- items : List
+ addItem(item : Item) : void
+ getTotal() : double
TransferBill
+ toJson() : String
Rappresentazione economica di un conto
Rappresentazione trasferibile attraverso servizio REST di un conto