

“Software Engineering”

Course

a.a. 2016-2017

Template version 1.0

Lecturer: Prof. Henry Muccini (henry.muccini@univaq.it)

Planner Path Calculator

version v2

Deliverables

Date	20/01/2017
Deliverable	<i>Deliverable 3</i>
Team (Name)	Indifferente

Team Members		
Name & Surname	Matriculation Number	E-mail address
Tommaso Di Salle	236202	Tommasodisalle@gmail.com
Luca D’Orazio	227635	Lucaadorazio@gmail.com
Stefano Corsetti	227288	s.corsetti@hotmail.it
Francesco Di Cato	203356	dicatofrancesco@gmail.com
Eugenio Mancini	230024	Emancini1992@libero.it

Index

- 1. List of Challenging/ Risk requirements or Task (pag.4)**
- A. Requirements Collection & Description (pag. 5)**
 - A.1 List Functional Requirements (pag.5-9)**
 - A1.1 GUI Requirements**
 - A1.2 Business Logic Requirements**
 - A1.3 DB Requirements**
 - A.2 Non Functional Requirements (pag.10)**
 - A.3 Content (pag.10)**
 - A.4 Assumption (pag. 10)**
 - A.5 Prioritization (pag. 11)**
 - A.6 UseCase Diagram (pag.12)**
 - A.7 Scenari (pag. 13)**
 - A.8 Descrizione UseCase (pag.14-19)**
- B. Analysis Model (pag.20-25)**
 - B.1 Robustness Diagram**
- C. Software Architecture (pag. 26)**
 - C.1 Component Diagram (pag.26-27)**
 - C.2 Sequence Diagram (pag. 28-35)**
- D. ER Design (pag. 36)**
- E. Class Diagram of the implemented System (pag.37-38)**
- F. Design Decision (pag.39-40)**
- G. FRs and the NFRs (pag.41-42)**
- H. Effort recording (pag. 43)**
- Code**

Table of Contents of this deliverable

-Nella Deliverable v1 abbiamo affrontato i punti A,B,C,F e H.

In questi punti si chiedeva di riportare la collezione dei requisiti (requisiti funzionali, scenari, assunzioni e prioritizzazioni) , l'Analisi dei modelli con il diagramma Robustness, l'Architettura software con il relativo diagramma di Componenti, le Design Decision ed infine una tabella con le ore spese per questo primo step d progetto.

-Nella Deliverable v2 abbiamo affrontato i punti A,B,C,D,E,F,G,H.

In questi punti, invece, si chiedeva di riportare, oltre alla revisione dei punti svolti nello step precedente, anche i requisiti non funzionali, ER Design, Class Diagram, Sequence Diagram e un prototipo della nostra applicazione.

-Nella Deliverable v3 abbiamo affrontato tutti i punti richiesti.

Oltre alla ristrutturazione di tutti i diagrammi, la relativa documentazione e la copia del codice, abbiamo montato dei video per far vedere il funzionamento della nostra applicazione rispettando i requisiti.

2.List of Challenging/Risky Requirements or Tasks

Challenging Task	Date the task is identified	Date the challenge is resolved	Explanation on how the challenge has been managed
Utilizzo Software MagicDraw	8/11/2016	8/11/2016	Consultazione manuale online poichè non trattato nello specifico a lezione
Dove eseguire calcolo: database o server?	2/12/2016	6/12/2016	Discussione in gruppo
Linguaggio di programmazione da utilizzare	5/12/2016	17/12/2016	Ricerca da fonti varie su web
Tipo di Database da utilizzare	5/12/2016	3/01/2017	Discussione di gruppo e ricerche
Connessione Client-Server ed invio dati	5/12/2016	23/12/2016	Ricerca da fonti varie su web
Struttura dati Albero	2/12/2016	2/12/2016	Il team ha operato con struttura ad array
Architettura Software	22/11/2016	20/01/2017	Approfondimenti su libro ti testo, slide ed internet su Component Diagram.
Creazione algoritmo per il calcolo del percorso	7/01/2017	9/01/2017	Facendo testing sul codice
Document/ Documentazione	18/01/2017	19/01/2017	Il team ha documentato tutte le scelte fatte in linguaggio natural

A. Requirements Collection

A.1 Functional Requirements

-Lista Requisiti funzionali

- 1- Creazione Albero
 - 1.1-inserimento parametri
 - 1.2- Salva Albero
- 2- Seleziona Albero
 - 2.1- Carica Albero
 - 2.2- Cancellazione
- 3- Calcolo Somma
 - 3.1- Scelta due vertici
 - 3.2somma tra i 2 vertici
- 4- tempo di calcolo

-Descrizione Requisiti funzionali

I numeri (*) rispettano l'ordine della lista dei requisiti funzionali.

Dall'analisi dei requisiti e dal file condiviso abbiamo estrapolato i seguenti R.F. che possono essere raggruppati in :

Creazione, caricamento, cancellazione, seleziona e calcolo.

Creazione(1) dedotto da **“The PPC must have a GUI that generates a tree structure using the following parameters”**.

SalvaAlbero(1.2) :L'albero verrà generato e salvato cliccando sul bottone “genera”. Requisito dedotto da “The GUI can be used to generate a tree when the “Build Tree” button is pressed. The result is saved in the database.”

Carica Albero(2.1): l'utente può scegliere un albero precedentemente salvato. Dedotto da “The GUI can be also used to retrieve from the database a Tree previously stored”

Cancellazione(2.2) dedotto dal file id domanda 6 con risposta “L'albero può essere inserito, cancellato e visualizzato. Non serve modificarlo.”

Seleziona(2): dedotta dal file id domanda 6(“L'albero può essere inserito, cancellato e visualizzato. Non serve modificarlo.”) e da id domanda 22(“Non è necessario visualizzare l'albero in quanto stiamo parlando di alberi potenzialmente grandissimi. Se trovate un modo efficiente di farlo va benissimo, ma consideratela una cosa opzionale. Quello che è richiesto invece è visualizzare il risultato dell'operazione di calcolo con la lista dei vertici attraversati (es: Vertice3, Vertice4, Vertice 5. Ptime=50 Cost=210)”). Abbiamo quindi deciso di inserire una funzionalità antecedente quella di caricamento e cancellazione in cui l'utente avrà una descrizione generale dell'albero(altezza, splitsize, numero totale nodi etc,) e da questa potrà scegliere se cancellare o modificare l'albero.

CalcoloSomma(3): che raggruppa Scelta di 2 vertici (3.1) e somma tra i 2 vertici(3.2) dedotto da “users can select **2 Vertices A and B, and the system must return the list of vertices from A to B among with the SUM of each attribute**

Tempo di calcolo(4) dedotto da “The GUI will also display the time the system took to make this calculation” **.

** per tempo di calcolo non intendiamo che il Sistema deve essere veloce a ritornare il risultato(NFR) ma che come funzionalità deve riportare a video il tempo impiegato

A1.1 GUI Requirements

La gui dovrà provvedere a fornire le seguenti funzionalità:

- Generazione albero premendo il bottone "genera";
- Inserimento parametri su nodi e archi premendo rispettivamente i bottoni "AggiungiAttrNodo" e " AggiungiAttrArco" ;
- Scelta dei due nodi (Nodo1 e Nodo2) con due input;
- Calcolo Somma premendo il pulsante "Calcola";
- Cancellazione Albero premendo il bottone "Cancella";
- Bottone "Lista" per visualizzare l'elenco degli alberi presenti nel database
- Bottone "Seleziona" per avere avere informazioni sull albero selezionato con il bottone checkbox
- Definito dall'utente un percorso (Nodo(i) e Nodo(j)) mi fa vedere la lista dei nodi coinvolti nell'operazione
- Se la lista degli alberi si viene rindirizzati direttamente nella pagina di creazione.
- la visualizzazione dell'albero in lista non avviene fintanto che l'albero non è stato completamente memorizzato
- Se si elimina un albero la pagina a cui si viene riportati è "Lista" a patto che ci sia almeno un albero presente

-Questa è la form di creazione

CREAZIONE ALBERO

DATI ALBERO

nomi:

Nome Albero

Nome Vertice

specifiché:

Split Size

Depth

GENERA

attributi nodi:

Nome Attributo Nodo

valoreMinNodo

valoreMaxNodo

aggiungiAttrNodo

attributi archi:

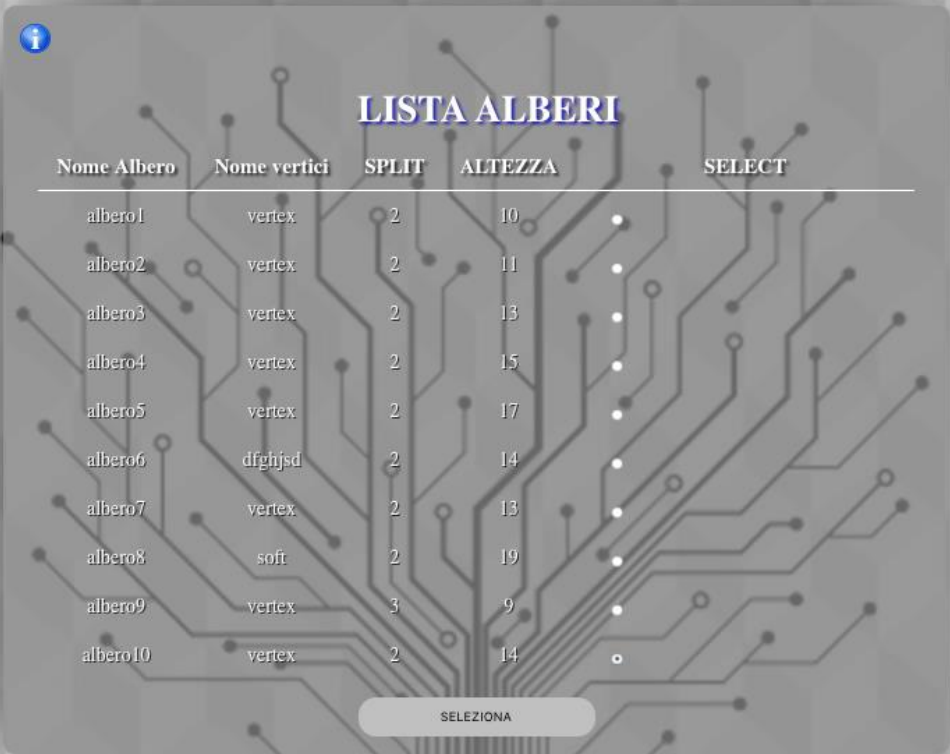
Nome Attributo Arco

valoreMinArco

valoreMaxArco

aggiungiAttrArco

-Qui troviamo la lista degli alberi creati

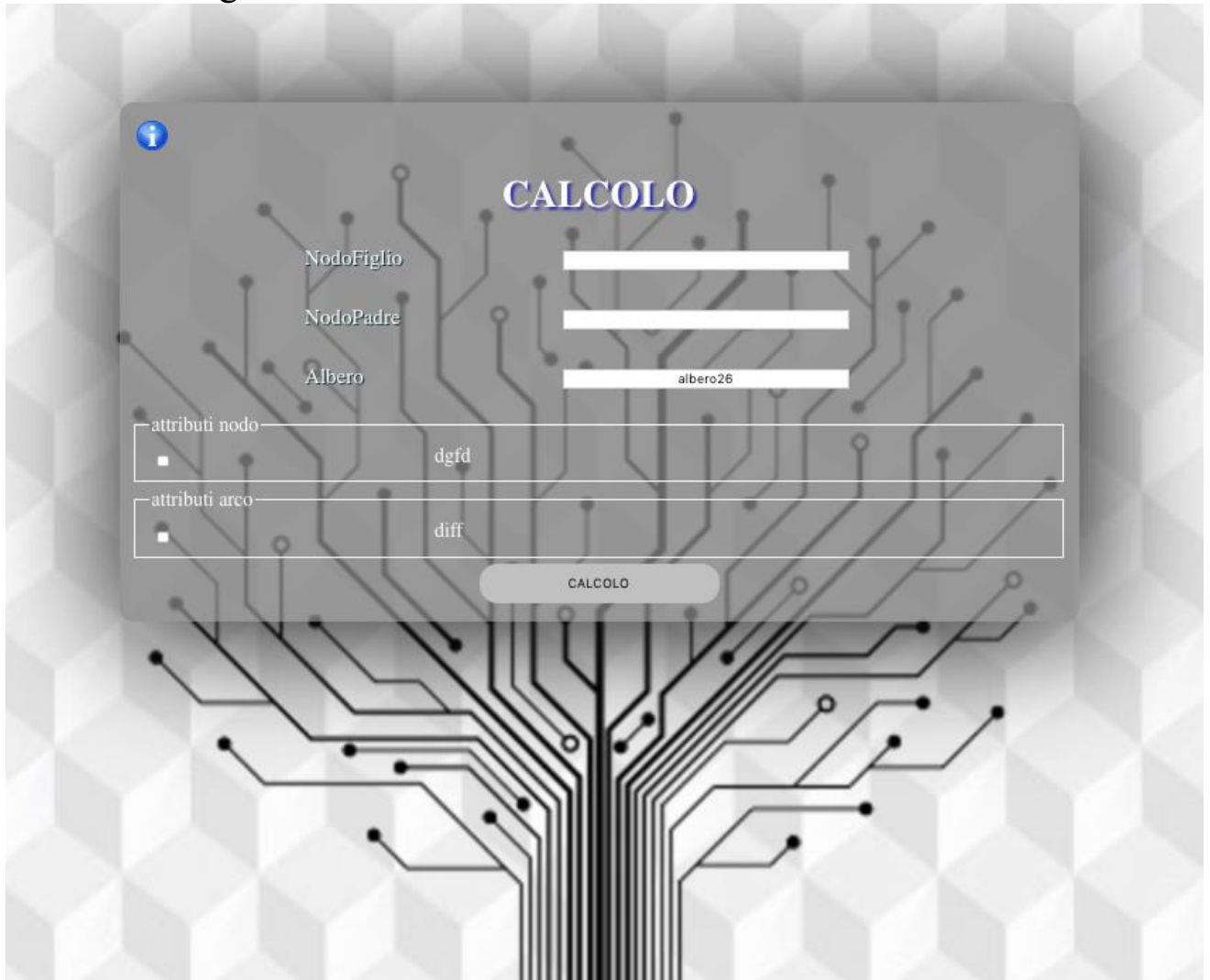


LISTA ALBERI

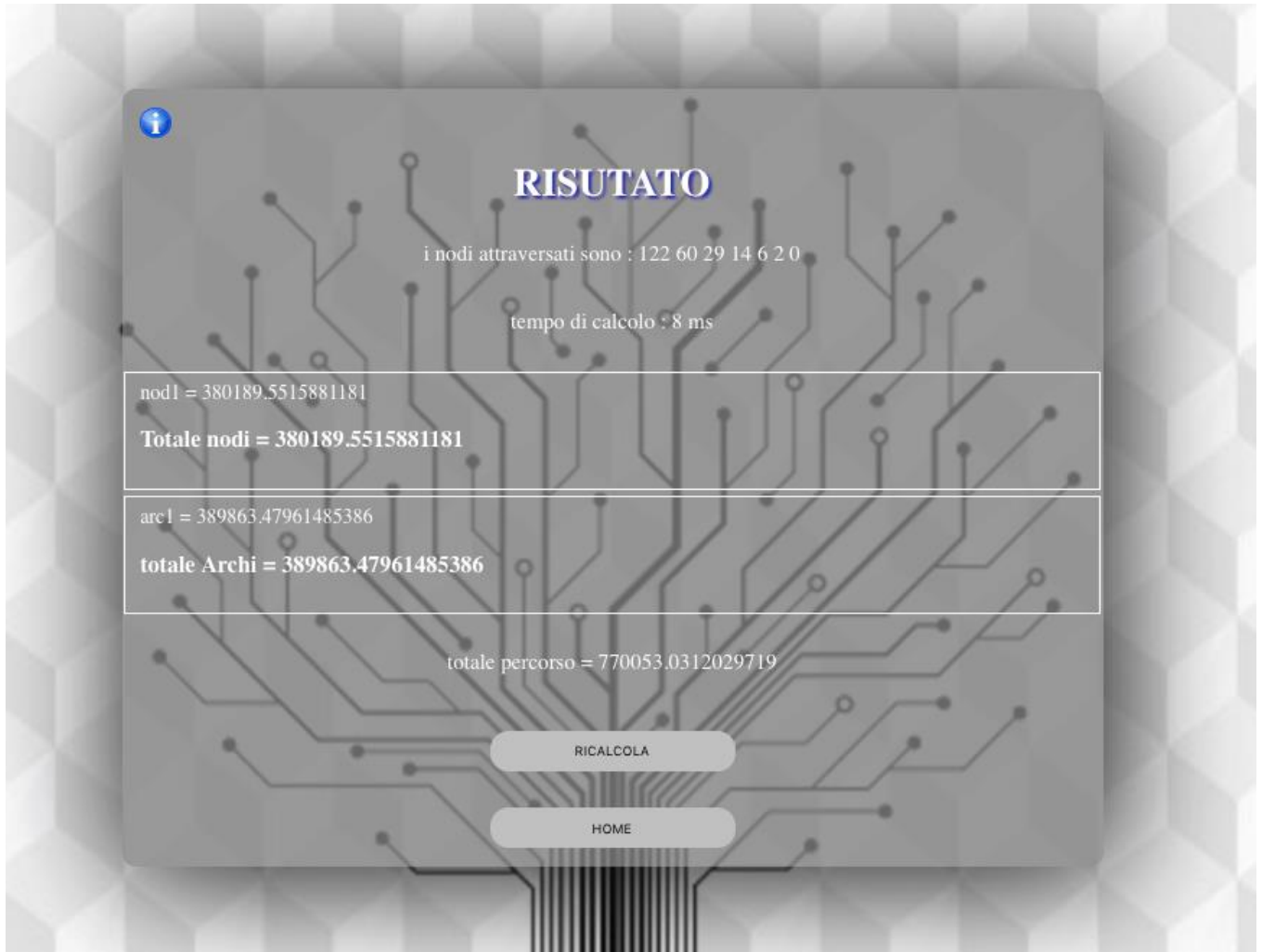
Nome Albero	Nome vertici	SPLIT	ALTEZZA	SELECT
albero1	vertex	2	10	<input type="radio"/>
albero2	vertex	2	11	<input type="radio"/>
albero3	vertex	2	13	<input type="radio"/>
albero4	vertex	2	15	<input type="radio"/>
albero5	vertex	2	17	<input type="radio"/>
albero6	dfghjkd	2	14	<input type="radio"/>
albero7	vertex	2	13	<input type="radio"/>
albero8	soft	2	19	<input type="radio"/>
albero9	vertex	3	9	<input type="radio"/>
albero10	vertex	2	14	<input type="radio"/>

SELEZIONA

-Questa è la schermata dove l'utente effettua il calcolo selezionando gli attributi interessati



-Infine abbiamo la schermata risultato dove è possibile vedere i nodi attraversati e le relative somme



A1.2 Business Logic Requirements

La Business Logic svolgerà le funzioni di:

- Creare un albero secondo i parametri inseriti in input dall'utente;
- selezionare un albero tra quelli presenti
- Cancellare un albero presente nel db;
- effettuare il calcolo della somma ed il tempo impiegato per calcolarla, degli attributi selezionati dal nodo di partenza al nodo di destinazione e riportare l'elenco dei nodi percorsi

A1.3 DB Requirements

- Il DB deve essere sempre accessibile e mantenere le informazioni precedentemente memorizzate

A.2 Non Functional Requirements

- Il Sistema deve essere veloce a ritornare i risultati;
- Gestire 10/100 utenti senza subire cali di prestazioni;
- Gestire alberi fino a 2000000 di nodi;
- Gui in html5;
- La sicurezza non deve essere garantita;

A.3 Content

- 1) I dati verranno immessi dall' utente per creare un nuovo albero.
- 2) Per recuperare un albero, i dati verranno estratti da un database.

A.4 Assumptions

- 1) Assumiamo che il Sistema sia collegato in rete.
- 2) Assumiamo che il Sistema disponga di risorse di calcolo adeguate.
- 3) Assumiamo che l'utente non utilizzi le funzionalità del browser (back, f5,..etc)

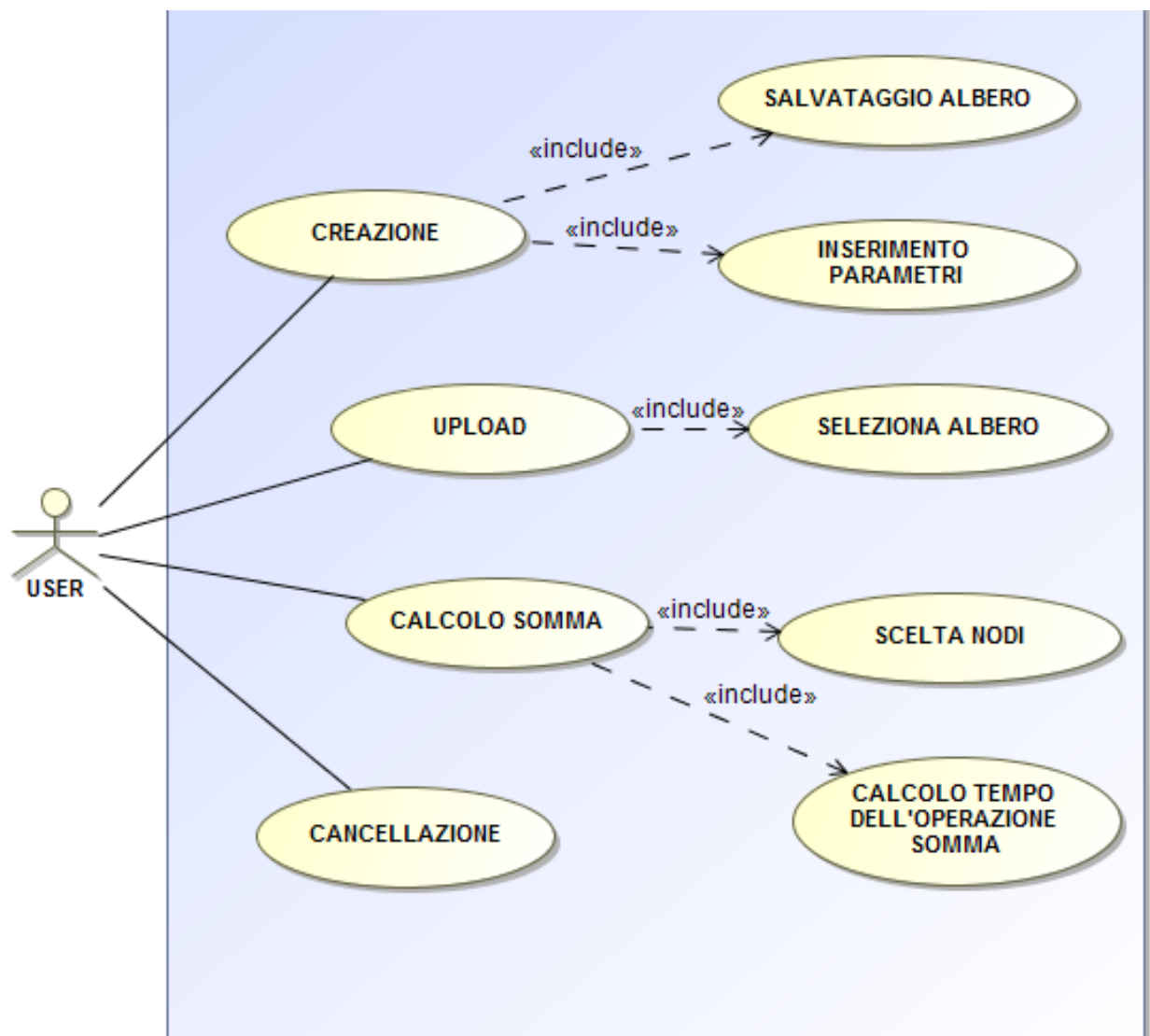
A.5 Prioritization

Requisiti listati secondo ordine di priorità.

Ordine Priorità	Requirement	Priorità	Motivazione
1	CalcoloSomma	High	Funzione indispensabile del sistema.
2	Creazione Albero	High	Senza questa funzionalità, si creerebbero alberi con poca utilità per l'utente.
3	InserimentoParametri	Medium	Funzionalità indispensabile richiesta dal cliente.
2	Caricamento	Medium	È necessario per eseguire altre funzionalità. Permette di caricare alberi già presenti nel sistema,
4	Cancellazione	Low	È una funzionalità opzionale, poiché senza di essa è comunque possibile effettuare le altre funzionalità (operazioni).

A.6 USE-CASE Diagram

Abbiamo deciso di fermarci ad un alto livello di astrazione. Abbiamo identificato solo i 4 use-case principali che sono emersi dai vari scenari ipotizzati e alcuni di questi sono stati esplosi con le funzionalità che riteniamo più importanti.



A.7 Scenari :

Di seguito alcuni possibili scenari:

1- CREA:

l'utente inserisce i parametri dell'albero (nomi, split-size e altezza).

Sceglie la lista di attributi da inserire a nodi ed archi ed il range di valori da assegnare ad ogni attributo. Cliccando sul pulsante GENERA verrà generato l'albero con tali proprietà e salvato.

2- SELEZIONA:

l'utente seleziona l'albero da un elenco presente nella lista e avrà informazioni sull' albero

3- CALCOLO SOMMA:

l'utente sceglierà 2 vertici. Tra questi 2 vertici verrà calcolata la somma di tutti gli attributi su archi e nodi e il risultato verrà stampato a video insieme al tempo impiegato per effettuare l'operazione

NOTA :

Qui è stata messa come funzionalità anche il tempo dove intendiamo il calcolo del tempo che il sistema impiega per calcolare la somma

4- CANCELLAZIONE

L'utente cliccando sul bottone "cancella" elimina l'albero selezionato dall'elenco

A.8 UseCase Description

USE CASE	Creazione	
Goal in Context	Generazione e Salvataggio Albero	
Scope & Level	L'utente sceglie i valori da assegnare all'albero e quest'ultimo viene salvato	
Preconditions	Inserimento Parametri corretti	
Success End Condition	L'albero viene generato	
Failed End Condition	(Notifica di errore)	
Primary Actor	USER	
Trigger	Click sul pulsante "GENERA"	
DESCRIPTION	STEP	ACTION
	1	L'utente inserisce i parametri (split-size, altezza, nome, lista attributi nodi/archi e range valori per ogni attributo)
	2	Salvataggio albero generato

RELATED INFORMATION	Creazione
Priority	High
Performance	< 2 MIN
Frequency	1 volta a settimana per utente

USE CASE	Seleziona Albero	
Goal in Context	Selezione di un albero tra quelli presenti	
Scope & Level	L'utente cliccando su un albero tra la lista di quelli presenti recupererà le informazioni ad esso associate (numero nodi, altezza, lista attribute, etc..)	
Preconditions	L'utente ha cliccato su "LISTA"	
Success End Condition	è stato selezionato un albero	
Failed End Condition	(notifica di errore)	
Primary Actor	USER	
Trigger	Click su albero	
DESCRIPTION	STEP	ACTION
	1	Click per la selezione di un albero

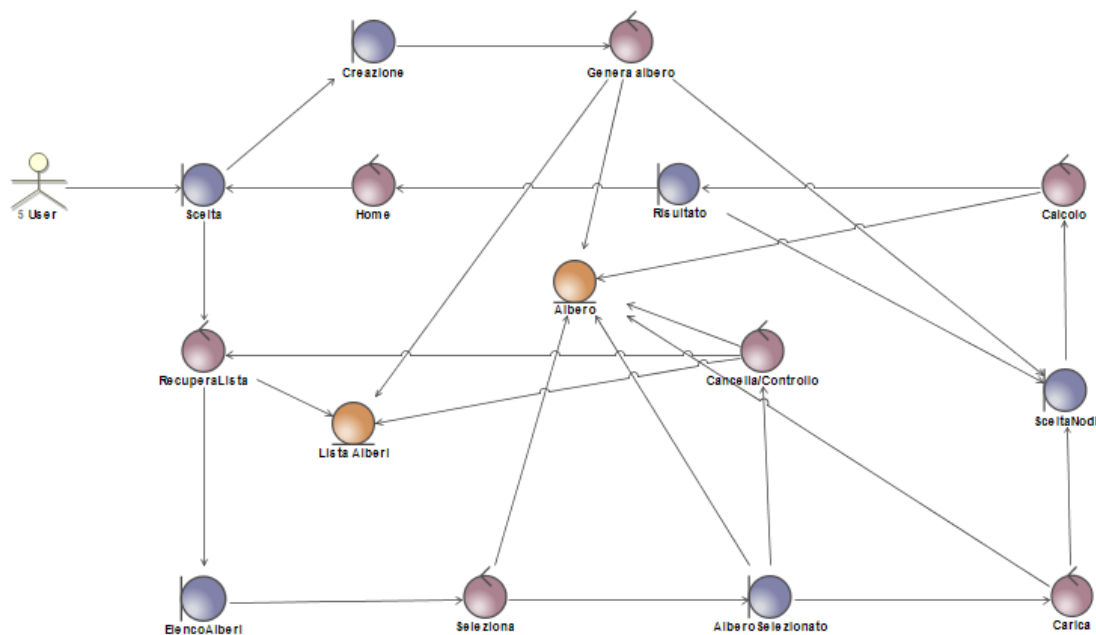
RELATED INFORMATION	Seleziona albero
Priority	Low
Performance	< 5 sec
Frequency	(caricamento) 3 volte al giorno per utente (cancellazione) 1 volta a settimana per utente
Superordinates	<ul style="list-style-type: none"> • Caricamento • Cancellazione

USE CASE	Calcolo Somma	
Goal in Context	Somma sugli attributi dei nodi selezionati	
Scope & Level	il Sistema fornirà la lista dei vertici tra i nodi selezionati e la somma dei valori degli attributi su nodi/archi di tale lista	
Preconditions	L'utente ha cliccato sul bottone "carica" o ha creato un nuovo albero	
Success End Condition	Stampa a video il risultato della somma e il tempo impiegato per calcolarlo	
Failed End Condition	Notifica Errore	
Primary Actor	USER	
Trigger	Click sul pulsante "CALCOLA"	
DESCRIPTION	STEP	ACTION
	1	L'utente sceglie due nodi
	2	Clicca sul bottone calcola
	3	Il sistema ritorna a video la lista dei vertici, il risultato della somma sui singoli attributi selezionati, su tutti i nodi/archi per ogni attributo selezionati, il totale della somma nodi+archi ed infine il tempo impiegato a calcolarla

RELATED INFORMATION	Calcolo somma
Priority	High
Performance	operazione immediata per qualsiasi dimensione dell'albero
Frequency	3 volte al giorno per utente
Subordinates	<ul style="list-style-type: none">• Scelta nodi• Tempo

B. Analysis Model

B.1 Robustness Diagram



Il team ha prodotto il diagramma di Analysis Model affrontando diverse decisioni.

Come boundary abbiamo identificato i seguenti oggetti:
Scelta, Creazione, Elenco Alberi, Albero selezionato, Scelta nodi, Risultato.

Come controller, abbiamo identificato i seguenti oggetti: Recupera lista, Seleziona, Carica, Cancella/Controllo, Calcolo, Genera albero, Home.

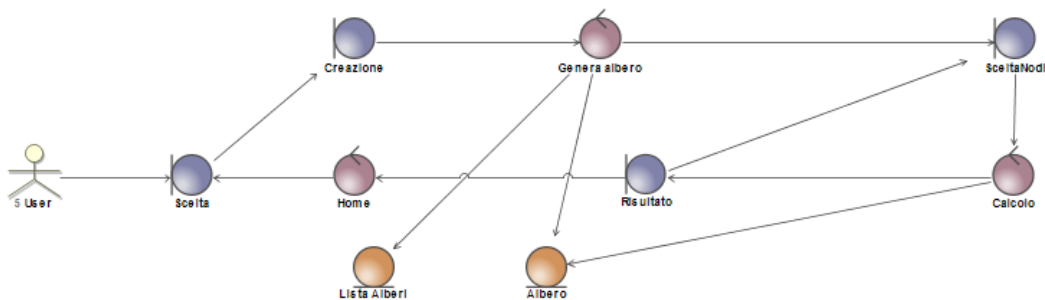
Come entity abbiamo identificato i seguenti oggetti: Lista alberi ed Albero.

Come primo step l'utente si interfaccia con una schermata, identificata dal boundary "Scelta", che rappresenta la pagina iniziale della nostra applicazione. L'utente avrà due opzioni: creare un nuovo albero cliccando sul bottone "Crea", oppure visualizzare la lista degli alberi presenti nel Database, cliccando sul bottone "Lista".

Ramifichiamo la spiegazione in base al bottone premuto e riportando il relativo diagramma.

Caso1

Bottone Crea:



-Form creazione: Boundary Object dove l'utente inserisce i parametri per la generazione dell'albero.

-GeneraAlbero: Controller che prende i parametri inseriti in FormCreazione e crea l'albero

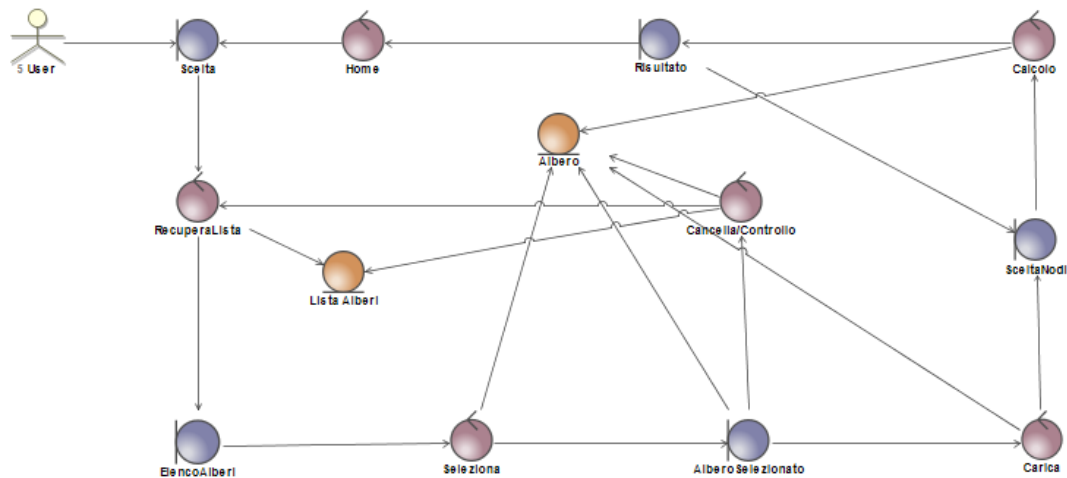
-
- descrizione funzionalità :

L'utente una volta generato l'albero può scegliere i nodi per il calcolo della somma. "SceltaNodi" va nel controller "Calcolo" per questo passaggio, per poi arrivare nel boundary "Risultato" dove potrà visualizzare il risultato atteso.

Adesso, da questa schermata (dove compare il risultato della somma) si può tornare nella scelta dei nodi (vedi collegamento tra i due boundary "Risultato" e "SceltaNodi") oppure tornare alla pagina iniziale (vedi collegamento tra Boundary "Risultato" con "home")

Caso2

Bottone Lista:



-RecuperaLista: l'utente, precedentemente ha cliccato su "Lista", questo Controller ha la funzionalità di recuperare le informazioni presenti in "ListaAlberi"

-ElencoAlberi: Entity che mostra l'elenco degli alberi presenti in "ListaAlberi". Cliccando sulla checkbox di un albero presente nella lista e cliccando sul pulsante "Seleziona" entra in funzione il controller "Seleziona"

-Seleziona: Controller che recupera informazioni sull'albero e si occupa delle concorrenza(guardare sezione DesignDecision)

-Albero Selezionato: Entity che mostra le informazioni dell'albero precedentemente selezionato. Avrà due bottoni, di cui uno "Carica" e uno "Cancella".

-Carica: Controller precedentemente citato, che ha la funzione di caricare l'albero

-Cancella: Controller che ha la funzione di eliminare definitivamente l'albero precedentemente selezionato

Object in comune tra i 2 percorsi:

-ListaAlberi: Entity Object in cui è inserita la lista dei nomi e attributi degli alberi presenti.

-Albero: Entity Object in cui è inserita la lista dei nodi ed il valore degli attributi

-SceltaNodi: Entity Object dove l'utente sceglie i due vertici e gli attributi su cui effettuare la somma

-Calcolo: Controller Object che ha la funzione di effettuare il calcolo della somma e l'elenco dei nodi tra i due selezionati

-Risultato: Boundary Object che mostra a video il risultato calcolato dal controller "Calcolo" ed il tempo impiegato per effettuare questa operazione.

Qui ci sono due bottoni "Home" e "Ricalcola".

Cliccando su "Ricalcola" l'utente viene portato nell'Entity "SceltaNodi" ed effettuare nuovamente calcoli sull'albero precedentemente selezionato

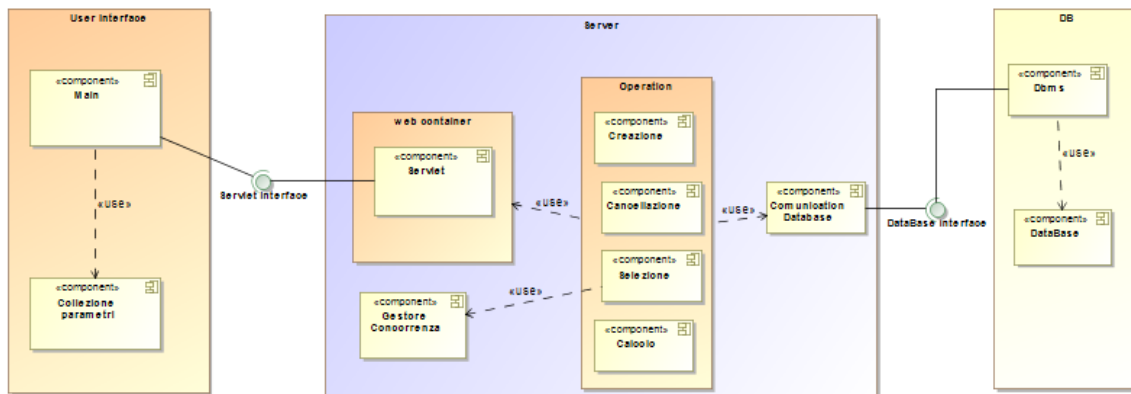
Cliccando su "Home" l'utente viene portato nell'Entity "Scelta" e si occupa della concorrenza (vedere nota DesignDecision)

- descrizione funzionalità :

Una volta selezionato l'albero ("Albero Selezionato") abbiamo due strade: una è la cancellazione IMMEDIATA di tale albero (recuperando l'albero che abbiamo identificato con l'entità "Albero" e quindi tornare alla schermata principale), mentre la seconda è caricarlo visualizzando le sue proprietà, per poi tornare alla sceltaNodi (vedi collegamento tra controller "Carica" e boundary "SceltaNodi") quindi effettuare il calcolo della somma su i nodi scelti.

C. Software Architecture

C.1 The static view of the system: Component Diagram



C1.1 Documentazione component diagram

Nel design del sistema abbiamo utilizzato un architettura client/server. La componente UserInterface(U.I.) richiede dei servizi alla componente Server che risponderà alle richieste interrogando la componente DB. Quindi ci sono 3 macro: U.I., Server e DB.

1) User Interface:

la component U.I. è l'interfaccia con cui comunica l'utente.

A sua volta è stata divisa in 2 component:

La prima è Main il quale si occupa di far visualizzare le pagine in cui l'utente non può inserire/ricevere dati. Mentre la seconda è Collezione Parametri, che si occupa di far visualizzare le pagine in cui l'utente inserisce/riceve dati dalla componente Server.

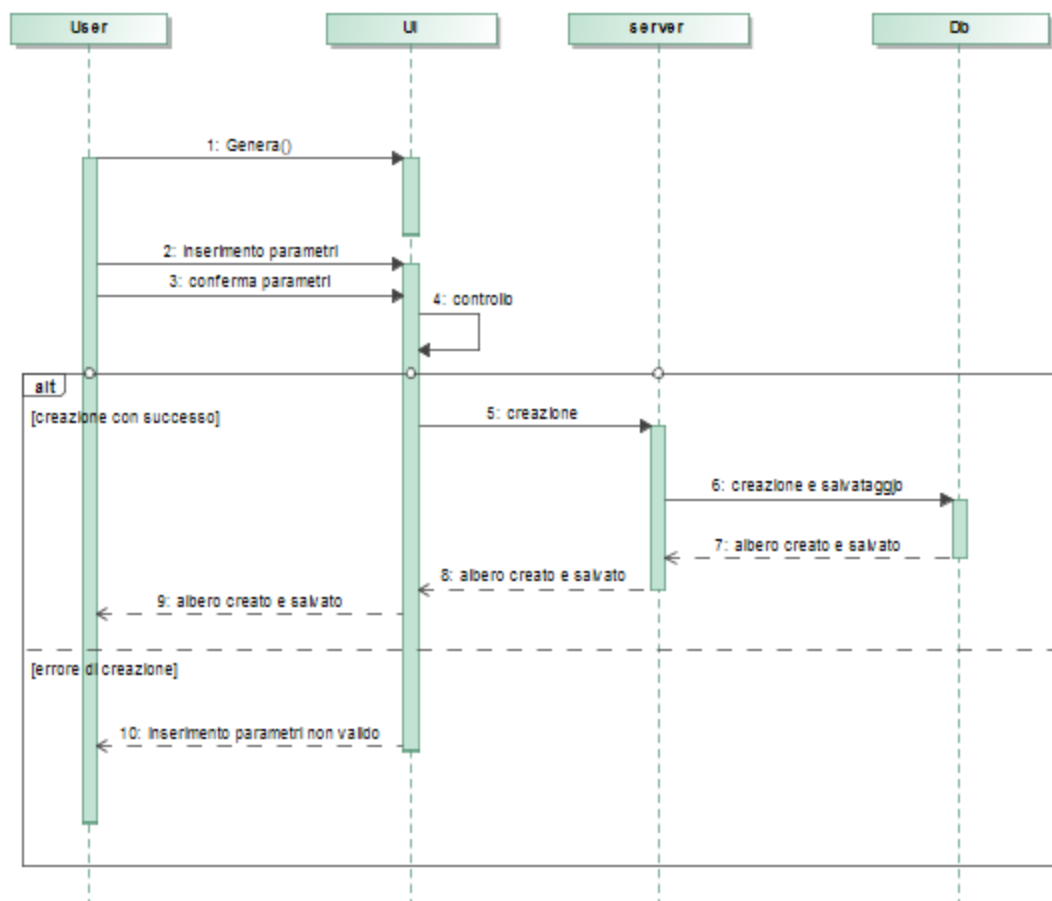
2) Il Server (rappresentante TOMCAT) è la componente che si occupa di servire le richieste della U.I. E' la parte principale del nostro sistema. E' stata divisa in 4 sottosistemi:

- 2.1) Operation, dove il team ha deciso di suddividere "operazioni" in quattro micro-componenti, rappresentati le quattro macro-funzioni della stessa quali la creazione dell'albero, il relativo caricamento, la cancellazione e l'esecuzione del calcolo della somma
- 2.2) Web Container è il componente che interagisce con servlet Java, quindi responsabile della gestione del ciclo di vita della servlet. E' quindi un contenitore che gestisce le richieste da parte della servlet creando istanze quindi crea e gestisce oggetti richiesta e risposta
- 2.3) CommunicationDatabase è il sottosistema che interroga il DB. Esso deve gestire eventuali race-condition nella creazione e cancellazione di un albero.
- 2.4) Gestore Concorrenza che si occupa della gestione della multiutenza per garantire l'utilizzo del sistema in parallelo

3) L'ultima componente DB sarà composta da un sottosistema dbms che si occuperà di gestire e recuperare i dati e una parte Database che sarà quella dove saranno realmente memorizzati.

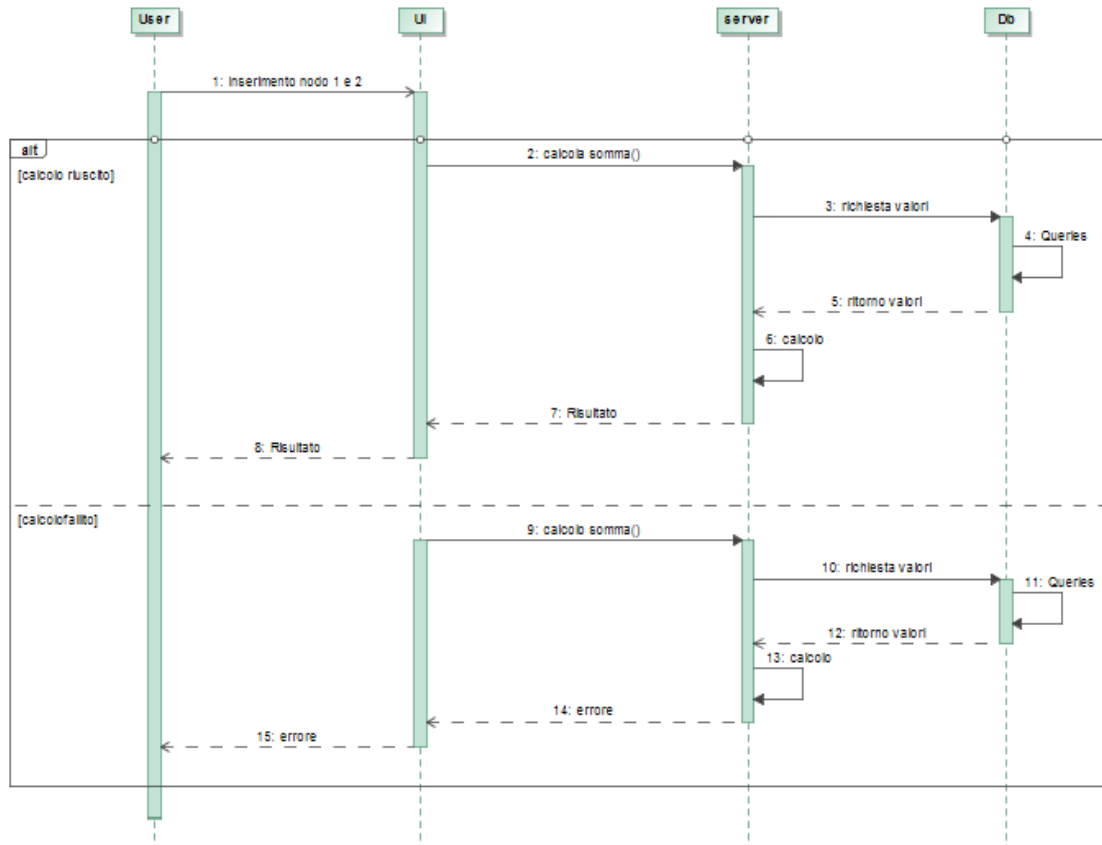
C.2 The dynamic view of the software architecture: Sequence Diagram

-Creazione albero



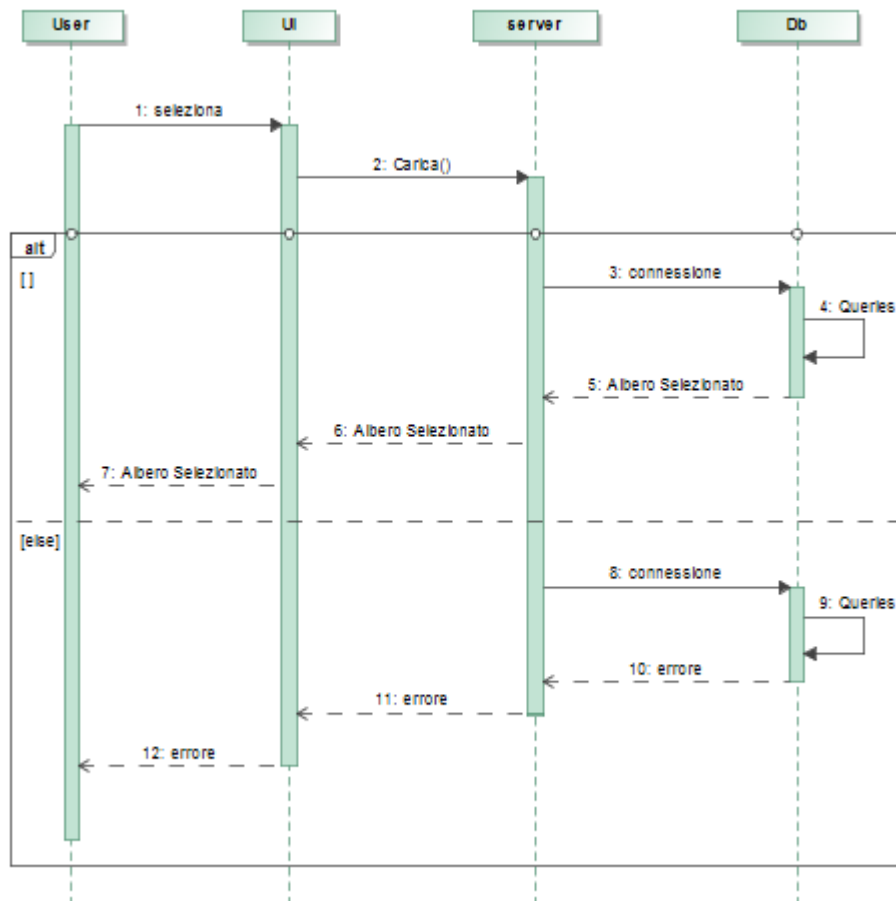
1. La creazione di un nuovo albero, generato a partire da una serie di dati inseriti dall'utente, mediante l'utilizzo della GUI e del suo codice interno.
2. Immissione dei parametri
3. Controllo correttezza
4. UI comunica al Server che è andato tutto a buon fine, quindi può generare l'albero
5. Server comunica al Database che è andato tutto a buon fine, quindi può creare e salvare l'albero
- 6,7,8. Creazione andata a buon fine quindi si può andare alla pagina di calcolo somma

-Calcolo su albero



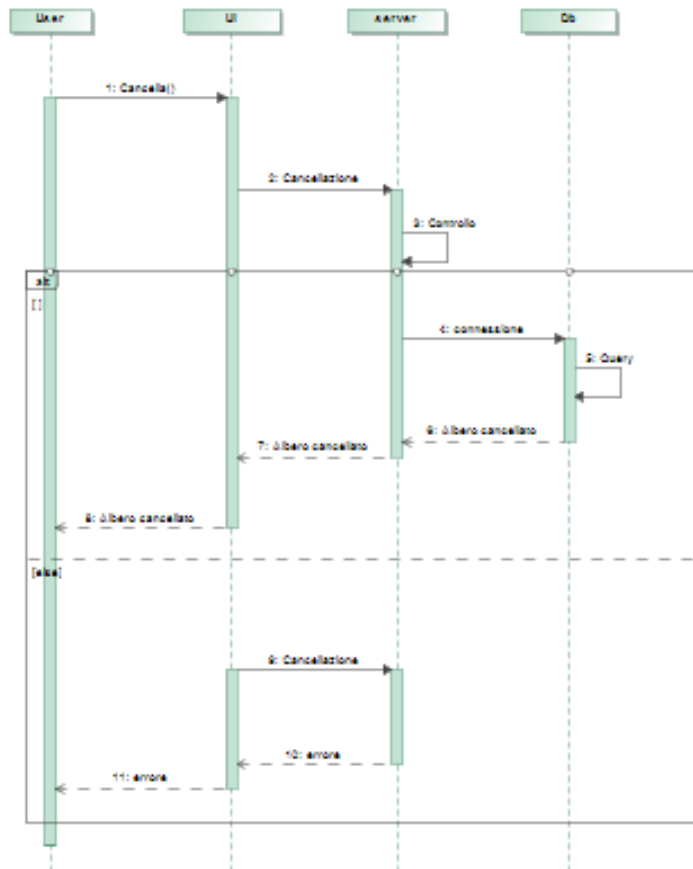
1. L'utente inserisce i parametri (Nodo1, Nodo2)
2. Pulsante di Calcolo della somma
- 3,4. Richiesta dei valori dei due nodi
5. Il server riceve le informazioni richieste
- 6,7 (if) Se i valori sono corretti il server effettua il calcolo della somma sui due nodi restituendo il risultato a schermo
8. UI rende disponibili il risultato dell'utente
- 9,10 (else) altrimenti restituisce un segnale di errore
- 11 UI rende disponibile il segnale di errore quindi dovrà ripetere la procedura di calcolo

-Caricamento Albero



1. L'utente seleziona un albero attraverso la UI
2. La UI avvia la procedura di caricamento di questo albero scelto nel database
3. (if) Il server si connette al database
4. Effettuando una query sul database è possibile riprendere le informazioni dell'albero
5. L'interfaccia con il database notifica il server dell'esito dell'operazione
6. Il server comunica l'esito alla UI
7. UI mostra l'esito dell'operazione all'utente
8. (else) il server non riesce a connettersi al database
- 9, 10, 11. Segnali di errori quindi dovrà ripetere la procedura di caricamento.

-Cancellazione



1. Si ha intenzione di cancellare un albero già presente
2. (if) UI comunica al Server l'intenzione di cancellare un albero
3. Il server effettua un controllo: verifica se l'albero È già utilizzato da altri utenti o meno
4. Il server si connette e quindi comunica con il database.....
5.che si preoccupa di eseguire una query relativa all'operazione di cancellazione
6. Database notifica al Server di aver cancellato l'albero
- 7,8. UI notifica l'utente la cancellazione dell'albero riuscita
9. (else) il server non si connette con il database
10. L'albero è già utilizzato da un altro/i utente/i, viene segnalato l'errore e quindi l'utente dovrà ripetere la procedura di cancellazione
11. Segnali di errori , quindi l'utente dovrà ripetere la procedura di cancellazione

D. ER Design

Il DB scelto è mongoDB, un database NoSQL.

Nel Database saranno inserite una collection chiamata “listaAlberi” che conterra l’elenco degli alberi presenti nel database con tutte le informazioni relative all albero scelte in fase di creazione:

_id : corrisponde al nome dell albero

NomeVertice: nome vertice scelto in fase di creazione

Split: Split selezionata

Altezza : depth selezionata

AttributoNodo[]: array contenente i nomi degli attributi del nodo

AttributoArco[]: array contenente i nome degli attributi dell arco

NumeroNodi: il numero totale dei nodi

E una collection per ogni albero presente in listaAlberi chiamata con il nome dell albero. Queste collections saranno così formate:

_id: numero del nodo

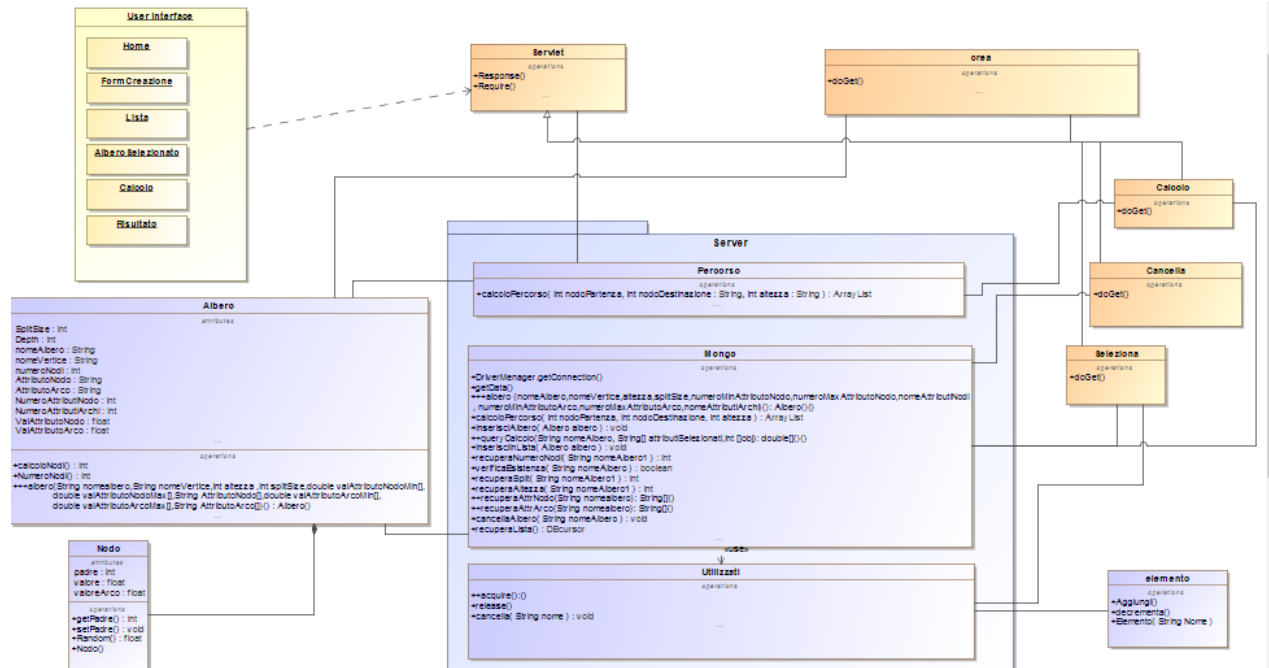
E una coppia chiave valore per ogni attributo di nodo e arco

La chiave corrisponde al nome dell attributo , il valore al valore generato random dal server.

Il nodo radice è memorizzato con _id : 0.

Il nodo radice ha comunque attributi per archi ma il valore è = 0 .

E. Class Diagram of the implemented System



-Descrizione class diagram

Class ALBERO:

Il team ha deciso di inserire questa classe nel diagramma poichè ritenuta fondamentale al funzionamento del server.

La classe albero contiene tutti gli attributi selezionati dall'utente in fase di creazione

Class Nodo:

Questa classe contiene il valore degli attributi del nodo e il valore dell'attributo dell'arco.

Class Mongo:

E' la classe che si occupa del collegamento e interrogazione con il database

Class Percorso:

Si occupa di calcolare a runtime i nodi inseriti dall'utente

Class Utilizzati/Elemento:

Ci permette di gestire la concorrenza. Ogni utente appena accede recupera un ArrayList se esiste, altrimenti viene generato. Ogni utente nell array list andrà ad inserire un oggetto della classe elemento che verrà inizializzato. Il nome dell'albero selezionato come stringa e un contatore che viene incrementato ogni volta che un utente seleziona l'albero.

Class Servlet:

La classe Servlet è una componente essenziale del sistema, in quanto grazie alle sue funzioni request() e response() ci permette di scambiare dati con la User Interface. La funzione request ci permette di prendere dati input, mentre response ci permette di restituire il risultato.

Le classi Calcolo, Creazione, Cancella e Seleziona:
sono un'estensione delle Servlet. Queste nel sistema hanno singole responsabilità, e per questo il team ha deciso di rappresentarle in questo modo.

La classe User Interface contiene tutte le classi che rappresentano le pagine con cui l'utente dovrà interagire.

F. Design Decisions

Di seguito vengono riportate le 5 più importanti design decision :

- 1 Utilizzare il minor numero di risorse HW /Creazione
- 2 velocità operazione calcolo
- 3 Collegamento Client/Server
- 4 Gestione Concorrenza
- 5 Scelta Database e scalabilità

1) Utilizzare il minor numero di risorse HW/ Creazione

Non disponendo di server e volendo sviluppare un app “testabile”, il team ha cercato di creare una web-app che potesse girare sulle nostre macchine. Il maggior numero di risorse sono richieste in fase di creazione. Dopo un primo studio si era pensato di utilizzare come struttura dati un array ,la struttura più efficiente in termini di risparmio di memoria ma qualsiasi tentativo, provato o ipotizzato, richiedeva un consumo eccessivo di memoria primaria, che ci portava ad un crash dell'applicazione dopo la generazione di 3 alberi con numero di nodi della dimensione massimale. Il problema non era relativo alla struttura dati utilizzata ma alla scarsa disponibilità di ram dei nostri calcolatori.

L'idea dell' utilizzo di una struttura dati era necessaria per come erano stata pensata l'operazione di calcolo. La Struttura dati serviva per la generazione di una gerarchia tra i vari elementi.

Trovata soluzione alternativa (vedi punto 2) non eravamo più vincolati all' utilizzo di una struttura dati.

Per questo il team ha deciso non utilizzare una struttura dati per memorizzare completamente l'albero e di memorizzarlo a run-time.

Questa design decision ci ha costretti a ridimensionare la generazione dell'albero:

Invece di creare una struttura dati dalla dimensione del numero di nodi il team ha pensato di spezzare tale valore in più parti in modo da limitare il numero di risorse consumate, e di non creare una classe arco ma dato che 2 nodi sono collegati da un solo arco considerarla come attributi della classe nodo.

N.B: non è la soluzione ottimale. Aumentando tale valore o utilizzando strutture dalla dimensione completa i tempi di creazione si riducono ma il consumo di risorse ,come spiegato precedentemente , aumentano drasticamente. La soluzione ideale trovata consisteva nella generazione completa di un albero con una struttura dati e successivamente utilizzare 2 thread. Uno che si occupasse di salvare l'albero nel DB e uno che permettesse all utente di effettuare l'operazione di calcolo sulla struttura presente in memoria. Con questa soluzione "simulavamo" tempi di creazione pari a zero ma il consumo di risorse è eccessivo.

2) Velocità operazione calcolo

Il team ha cercato in tutti i modi di rendere il più efficiente possibile l'operazione di somma

Soluzioni provate/scartate

-1 mantenimento del padre: per ogni nodi veniva inserito un riferimento a l'id del padre. Questa soluzione comportava l'interrogazione del DB più volte (una per ogni nodo per poter risalire al nodo antenato)

Nel Worstcase split = 2 l'operazione di calcolo richiedeva 21 interrogazioni.

-2 mantenimento dell'intero percorso: per ogni nodo mantenevamo una stringa con l'intero percorso. Per l'operazione di calcolo necessitavamo di interrogare il db 2 volte, una per il recupero del percorso e uno per reperire il valore degli attributi degli altri nodi coinvolti.

In entrambe le soluzioni trovate il problema era relativo al recupero del percorso.

Soluzione adottata Il team ha scritto un algoritmo che calcola a runtime il percorso. Così facendo utilizziamo una sola interrogazione al db per reperire i risultati (vedi descrizione algoritmo)

3) Collegamento Client/Server

Per essere sicuri di rispettare i vincoli imposti dal cliente il team ha deciso di affidarsi ad un web server che gestisse le connessioni tra utenti e server e di utilizzare le servlet per lo scambio di informazioni

4) Gestione Concorrenza

Il team ha identificato le seguenti problematiche di concorrenza

-In fase di Creazione

-In fase di cancellazione

In fase di Creazione il team ha sfruttato l'atomicità delle operazioni del db, senza dover creare nuove classi.

Il primo utente che crea un albero crea una collection con il nome dell'albero.

Se un secondo utente cerca di effettuare una creazione con il medesimo nome l'operazione fallisce.

In fase di cancellazione bisognava permettere a più utenti di effettuare l'operazione di somma in contemporanea, mentre l'operazione di cancellazione è possibile solo se nessuno sta effettuando operazioni sull'albero selezionato.

Per risolvere questo problema il team ha dovuto utilizzare i semafori per garantire la mutua esclusione e di una nuova struttura dati che mantenesse in memoria gli alberi attualmente utilizzati.

5) Scelta Database & scalabilità

Inizialmente l'app era stata sviluppata con mysql con query ad hoc e la generazione di tabella a run time per la memorizzazione di un albero . Questa soluzione creava diversi problemi quando simulavamo la creazione di diversi alberi con più utenti. I tempi di inserimento crescevano esponenzialmente. Per questo il team ha preferito cambiare database e utilizzare un nosql.

Il team ha scelto come Database MongoDB.

Questa scelta è dovuta proprio alle elevate performance nelle CRUD operations anche (soprattutto) con grandi quantità di dati e alla capacità di mongoDB di scalare orizzontalmente.

Per la scalabilità si rimanda il cliente al sito ufficiale di mongoDB.(esula dallo sviluppo dell app

configurare più istanze del BD ma la scelta di mongoDB è stata fatta anche per questo motivo)

G. Explain how the FRs and the NFRs are satisfied by design

Per quanto riguarda i vincoli funzionali vedere sezioni precedenti. Per quanto riguarda i vincoli non funzionali Di seguito elencati verranno analizzati uno per uno

-Il Sistema deve essere veloce a ritornare i risultati;

l'algoritmo che esegue il calcolo con interrogazione del db ridotta ad una query è efficiente. Il risultato torna sempre al di sotto del secondo in condizioni estreme e pochi decimi in condizioni ottimali

-Gestire 10/100 utenti senza subire cali di prestazioni; Per come è stata disegnata e implementata l'applicazione regge il numero delle utenze. Il data base è configurato per gestire 10 mila connessioni contemporaneamente e il web server per quantità anche maggiori. Prove in simultanea per il calcolo non è possibile farle poiché il calcolo impiega pochissimo tempo(guardare video2 per avere idea della gestione delle multiutenza con alberi di dimensione massima e superiore)

Cali di prestazioni nell'operazione di calcolo non ci sono, mentre nell'operazione di creazione cresce linearmente con la dimensione dell'albero e il numero di attributi inseriti

-Gestire alberi fino a 2000000 di nodi, l'applicazione è stata provata anche con alberi più grandi di 2milioni, prove sono state fatte con split size 2 e depth 25

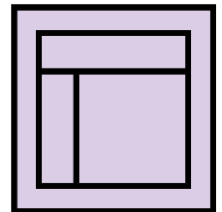
l'applicazione è stata testata con alberi anche più grandi di 2milioni

-Gui in html5;

html5 e più in generale html è un linguaggio di markup(è solo di rappresentazione) le pagine generate sono tutte in html e jsp(java server page : pagine html in cui è possibile iniettare codice java)

H. Effort Recording

GANTT



indifferente					
Corsetti Stefano, Di Salle Tommaso, D'Orazio Luca, Mancini Eugenio, Di Cato Francesco					
5					
s.corsetti@hotmail.it , tommasodisalle@gmail.com , lucaadorazio@gmail.com , Emancini1992@libero.it , dicatofrancesco@gmail.com					
When (Month/Day)	Time spent	Partners (please report how many people have been working)	Brief Description of the performed task	Category	Sub-Category
12/28	3	2	Collegamento da server a database NoSql	doing	Code-implementation
1/6	1	1	esecuzione calcolo	doing	code-implementation
1/17	3	1	Revisione diagrammi primo e secondo step	doing	diagram
1/18	4	2	Documentazione e revisione	doing	diagram
12/27	2	5	Decisioni su implemetazione	doing	code-implementation
1/9	3	2	Concorrenza con uso di Semafori	doing	code-implementation
1/15	5	2	Grafica Definitiva	doing	code-implementation
1/4	4	2	Gestione errori in fase di inserimento		
TOTALE ORE SPESE :		79			

Appendix. Code

In allegato con la deliverable viene consegnato il file .war e il progetto completo zippato per non riportare tutto il codice e appesantire troppo la deliverable (circa 2 mila righe di codice).

Di seguito verranno riportati solo gli algoritmi piu significativi: Calcolo, Creazione.

1) calcolo

```
public ArrayList calcoloPercorso(int nodoPartenza,int nodoDestinazione,int altezza){

    ArrayList<Object> percorso= new ArrayList();

    percorso.add(nodoDestinazione);

    if(nodoPartenza==nodoDestinazione)return percorso;

    calcoloPercorso(percorso, nodoPartenza, nodoDestinazione,altezza);

    for(int f=0;f<percorso.size();f++){

        if((int)percorso.get(f)==nodoPartenza)return percorso;

    }

    return percorso;

}

public ArrayList calcoloPercorso(ArrayList percorso,int nodoPartenza,int nodo,int altezza){

    if(percorso.size()>altezza)return percorso;

    if(nodo==nodoPartenza)return percorso;

    if((nodo%this.splitSize)!=0)return  calcoloPercorso(percorso, nodoPartenza, nodo+1,
altezza);

    nodo=(nodo/this.splitSize)-1;

    percorso.add(nodo);

    return calcoloPercorso(percorso, nodoPartenza, nodo, altezza);

}
```

Questo è l'algoritmo che ci permette di calcolare a run time il percorso attraversato dai due nodi scelti all'utente.

E' un algoritmo ricorsivo che sfrutta una proprietà che abbiamo riscontrato nell'albero: ogni nodo ha come ultimo figlio "(il suo numero x splitsize)+ splitsize" sfruttando questa caratteristica scorriamo tutti i figli fin quando non arriviamo all'ultimo figlio attraverso l'operatore modulo.

N.B: la gestione degli errori viene gestita semplicemente se il nodo di partenza è presente nel percorso restituito dall'algoritmo

```
public double [] queryCalcolo(String nomeAlbero, String[]
attributiSelezionati,int []obj){

    double []val=new double[attributiSelezionati.length];

    DBCollection collection= db.getCollection(nomeAlbero);

    DBObject query = QueryBuilder.start().put("_id").in(obj).get();

    DBCursor result = collection.find(query);

    while(result.hasNext()){

        BasicDBObject dbObject = (BasicDBObject)result.next();

        System.out.println(dbObject);

        for(int z=0;z<attributiSelezionati.length;z++){

            val[z]+= dbObject.getDouble(attributiSelezionati[z]);

        }

    }

    return val;

}
```

L'interrogazione è solamente una e consiste nel passare il percorso dell'algorithm precedente al database (da immaginare come una serie di || consecutivi)

2) Inserimento albero nel database (Creazione)

```
public void inserisciAlbero (Albero albero){  
  
    DBCollection coll = db.getCollection(albero.nomealbero);  
  
    List<DBObject> documents = new ArrayList<>();  
  
    Nodo                                nodoradice=new  
Nodo(albero.valAttributoNodo,albero.valAttributoArco);  
  
    DBObject documentRadice = new BasicDBObject();  
  
    documentRadice.put("_id", 0);  
  
    for(int g=0;g<nodoradice.valoreNodo.length;g++){  
  
        documentRadice.put(albero.AttributoNodo[g],  
nodoradice.valoreNodo[g]);  
  
        }  
  
    for(int g=0;g<nodoradice.valoreArco.length;g++){  
  
        documentRadice.put(albero.AttributoArco[g], 0);  
  
        }  
  
    documents.add(documentRadice);  
  
    for(int u=1;u<albero.numeroNodi;u++){  
  
        Nodo                                nodo=new  
Nodo(albero.valAttributoNodo,albero.valAttributoArco);
```

```
DBObject document1 = new BasicDBObject();

document1.put("_id", u);

for(int g=0;g<nodo.valoreNodo.length;g++){

    document1.put(albero.AttributoNodo[g],
nodo.valoreNodo[g]);

}

for(int g=0;g<nodo.valoreArco.length;g++){

    document1.put(albero.AttributoArco[g],
nodo.valoreArco[g]);

}

documents.add(document1);

if(documents.size()>1000){

    coll.insert(documents);

    documents.clear();

}

}

coll.insert(documents);

}
```

Questo è l'algoritmo che ci permette di inserire un albero al di sotto dei 30 secondi. L'algoritmo utilizza un arraylist che viene svuotato a raggiungimento di mille elementi

