Università degli Studi di Modena e Reggio Emilia

Dipartimento di Ingegneria "Enzo Ferrari"

Corso di Laurea in Ingegneria Informatica

Sviluppo di un'applicazione web per l'utilizzo del framework SparkER

Relatore:

Prof.ssa Sonia BERGAMASCHI

Correlatore:

Ing. Luca GAGLIARDELLI

Candidato:

Giovanni BONISOLI

Matr. 89698

Anno Accademico 2016-2017

SOMMARIO

o INTRODUZIONE

Nell'ambito delle attività di ricerca sui Big Data del gruppo di ricerca delle Basi di Dati, DBGroup (www.dbgroup.unimore.it) è stato sviluppato SparkER, un framework per l'Entity Resolution (ER). Con il termine: Entity Resolution o Record Linkage o Record Deduplication si indicano l'insieme di tecniche che permettono di rilevare istanziazioni dello stesso oggetto del mondo reale presenti in più sorgenti dati ed eventualmente riconciliarle se inconsistenti. SparkER è un framework di ER altamente scalabile, implementato sul middleware per la gestione dei Big Data SPARK. Per SparkER non era stata ancora sviluppata un'interfaccia grafica di supporto all'utente di ER.
L'obiettivo della mia attività progettuale è stato la realizzazione di un'applicazione web in grado di interfacciarsi con SparkER che consentisse ad un utente, anche non avanzato, l'utilizzo del framework. Era richiesto che l'applicazione permettesse di configurare SparkER, monitorare lo stato delle esecuzioni, analizzare e confrontare graficamente i risultati prodotti dai test. Inoltre, essa doveva essere accessibile anche da dispositivi mobile e in grado di inviare notifiche all'utente tramite il servizio di messaggistica Telegram. L'applicazione doveva, infine, essere in grado di funzionare sul computer ad alte prestazioni PICO di CINECA, sul quale non è possibile installare alcun DBMS, per cui i dati relativi ai test fatti dovevano essere mantenuti in modo alternativo, tramite dei file JSON.
Durante questa attività sono stato seguito dall'ingegnere Luca Gagliardelli, studente della scuola di dottorato in ICT dell'Università di Modena e Reggio Emilia e membro del team che ha sviluppato SparkER. Egli mi ha esposto nel dettaglio le specifiche del progetto e suggerito buona parte degli strumenti studiati e utilizzati.
Il presente elaborato ha lo scopo di illustrare il lavoro svolto, evidenziando le scelte progettuali, le caratteristiche degli strumenti usati e il risultato finale.

Il primo capitolo riguarda il framework SparkER, in particolare i parametri necessari per configurarlo e l'output che esso genera. Queste ultime caratteristiche sono state quelle che maggiormente hanno portato alla necessità di questa applicazione.

Il secondo capitolo riguarda la specifica dei requisi dell'applicazione.

Il terzo capitolo introduce le tecnologie usate per lo sviluppo dell'applicazione, partendo da una breve descrizione di: HTML, JavaScript e CSS, strumenti ormai indispensabili nella programmazione web lato client.
Viene poi introdotto Bootstrap, un framework che consente la creazione di siti web dotati di *responsive design* (adattabilità ai diversi dispositivi delle pagine web), e Highcharts, libreria per la creazione di grafici.
Successivamente viene descritto AngularJS, framework per lo sviluppo web lato client. Quest'ultimo ha determinato scelte progettuali importanti.
Viene poi introdotto brevemente il Python, linguaggio utilizzato per la parte server dell'applicazione, e, in particolare Flask, il framework scelto per implementare il server web.
Infine si fa un breve accenno a Telepot, una libreria di Python che serve per interfacciare un'applicazione conTelegram.

Nel quarto capitolo viene llustrato il risultato finale: è mostrata l'impostazione generale data all'applicazione e le sue principali funzionalità anche tramite il supporto di diagrammi UML.

Il capitolo Conclusioni riporta le note conclusive relative al lavoro svolto.

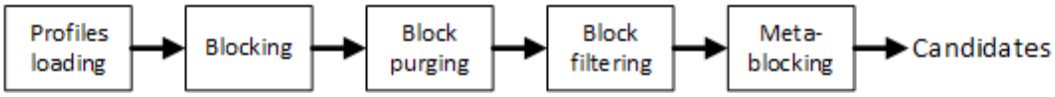
1 SPARKER

1.1 ENTITY RESOLUTION

L'integrazione dei dati è il processo che si occupa di fornire all'utente una visione unificata dei dati provenienti da diverse sorgenti e di diverso tipo. Si tratta di un trovare le corrispondenze tra i concetti rappresentati nelle sorgenti e fornire un'informazione globale su questi concetti o entità. L'Entity Resolution (ER) è uno step dell'integrazione dei dati che ha lo scopo di identificare e associare i record (profili di entità) provenienti da diverse sorgenti di dati (raccolta di entità) che si riferiscono alla stessa entità del mondo reale. Ciò costituisce un problema di complessità quadratica poiché richiede che ogni record sia confrontato con tutti gli altri. Ad un aumento lineare dei record corrisponde una crescita quadratica del numero dei possibili confronti, cosa che è computazionalmente insostenibile nel contesto dei Big Data dove si hanno milioni di records. Per ridurre questa complessità l'ER utilizza tecniche, dette tecniche di "blocking", che suddividono i profili in clusters chiamati blocchi. I confronti verranno effettuati solo tra profili all'interno di uno stesso blocco. Il numero di confronti rimane però troppo alto nel contesto dei Big Data, perciò vengono introdotte tecniche di "meta-blocking". L'idea è quella di ricavare un grafo in cui i nodi rappresentano i profili e gli archi rappresentano i confronti tra essi. Lo scopo è quello di stabilire delle proprietà metriche sul grafo e utilizzarle per eliminare gli archi meno significativi.

1.2 FUNZIONAMENTO DI SPARKER

SparkER segue il workflow presentato nella seguente immagine.



Prima di raggruppare i record è necessario individuare all'interno di essi i profili di entità. Si definisce profilo *p* di entità una tupla <*id*, A_p> dove *id* è un identificatore univoco e A_p è un insieme di coppie nome-valore <*n*,*v*> (o attributo-valore). Un insieme di profili è detto entity collections. Un dataset è di tipo "Dirty" se contiene al suo interno dei duplicati, altrimenti è di tipo "Clean". SparkER può ricevere in input un solo dataset Dirty in cui cercare i duplicati o due dataset Clean tra i quali cercare le corrispondenze.
La prima fase è quella di "profiles loading" (caricamento dei profili) che prevede di individuare i profili di entità all'interno di un dataset.

Si passa poi alla fase di blocking, in cui i profili vengono suddivisi in blocchi in base a un elemento detto "blocking key" (chiave di blocco). I confronti verranno effettuati solo tra profili all'interno dello stesso blocco. Esistono due tipi di raccolte di blocchi:

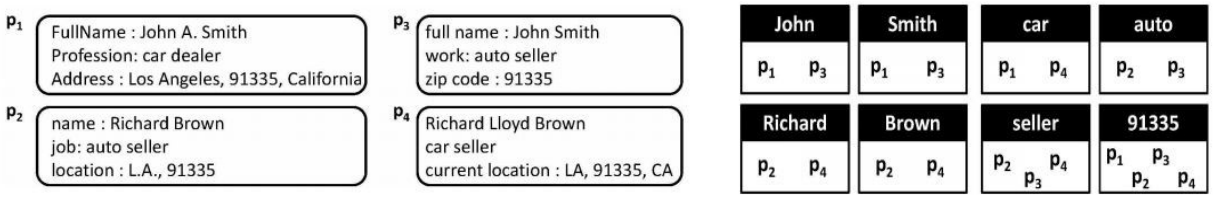
- unilaterale. Contiene profili di entità provenienti dalla stessa raccolta di entità, la quale può contenere dei profili corrispondenti tra di loro. I confronti devono essere fatti tra tutti i profili del blocco.
- bilaterale. Diviso in due sottoblocchi che all'interno non hanno profili corrispondenti fra di loro. I confronti, quindi, possono essere fatti solo con profili dell'altro sottoblocco.

SparkER implementa due possibili tecniche di blocking:

- "Token Blocking". Ogni token (valore) di un dataset viene considerato come chiave di blocco.

Vengono generati blocchi associati ad un token e in ogni blocco viene immesso ogni profilo in cui compare questo token, senza tener conto del nome dell'attributo associato ad esso.

- "Loose Schema Blocking". Vi è una fase preliminare di clusterizzazione degli attributi chiamata "Loose Attribute-match Induction". Questa tecnica genera dei clusters che contengono attributi simili in base ad alcuni parametri per il calcolo della similarità che vengono dati in input al programma. Potranno essere usati come blocking keys soltanto i token che compaiono in attributi considerati simili (o uguali).



A sinistra, un insieme di profili di entità. A destra, una serie di blocchi derivati dal Token Blocking.

La tecnica mostrata fa uso della ridondanza (un profilo può comparire in più blocchi), che è essenziale per evitare di perdere corrispondenze nel contesto di insiemi di dati eterogenei. Si perde però in efficienza in quanto rimane molto alto il numero dei possibili confronti da fare. Per mantenere l'efficacia della ridondanza e migliorare l'efficienza seguono ulteriori operazioni sui blocchi generati.

Tra queste fasi, vi è il "block purging" (purificazione dei blocchi), in cui vengono rimossi i blocchi 'sovradimensionati', ovvero blocchi che contengono un numero eccessivo di confronti; questo per aumentare l'efficienza e perché le loro blocking keys sono dei token molto comuni (come le "stopwords") e non forniscono corrispondenze significative. Bisogna definire un limite superiore riguardo la cardinalità dei blocchi in modo che quelli che lo superano vengano scartati senza un impatto significativo.

La fase successiva è quella del "block filtering", un'operazione che rimuove i profili dai blocchi in cui la loro presenza non è necessaria. Per ogni profilo, i blocchi che lo contengono (n blocchi) vengono disposti in ordine decrescente di importanza in base a un determinato criterio, ovvero la cardinalità di un blocco: meno confronti contiene un blocco, maggiore è la sua importanza. I blocchi vengono poi ordinati dal più piccolo al più grande. Viene determinato un limite massimo di blocchi per profilo, tramite la "filtering ratio" (r), un valore definito nell'intervallo [0,1] . Per ogni profilo vengono mantenuti gli n*r blocchi più importanti.
L'ultima fase è quella del meta-blocking, quella in cui i blocchi vengono ristrutturati in modo da tenere solo i confronti più promettenti. Prevede la costruzione di un grafo pesato che rappresenta la raccolta di blocchi ottenuta dopo la altre fasi. In questo grafo, i nodi rappresentano i profili di entità e gli archi esistono se i due nodi collegati compaiono insieme in almeno un blocco. Ogni arco è pesato secondo una determinata funzione di ponderazione che viene indicata da un parametro di input del programma. Vi sono sei possibili funzioni di pesatura in SparkER: "Aggregate Reciprocal Comparisons Scheme" (ARCS), "Common Blocks Scheme" (CBS), "Enhanced Common Blocks Scheme" (ECBS), "Jaccard Scheme" (JS) , "Enhanced Jaccard

Scheme" (EJS). Per ogni diverso metodo dato in input al programma verrà effettuato il meta-blocking. In base ai pesi degli archi viene effettuato il "pruning" (potatura), ovvero vengono tolti degli archi. Può essere di due tipi:

- "Weight Node Pruning"(WNP). Per ogni nodo viene calcolata una soglia locale o pari alla media dei pesi degli archi adiacenti (avg) o pari alla metà del massimo tra questi pesi (maxdiv2). Se il peso di un edge è maggiore di questa soglia allora viene mantenuto dal nodo, altrimenti no;
- "Cardinality Node Pruning" (CNP). Per ogni nodo viene calcolata una soglia locale che funge da massimo numero di archi da mantenere. Poi gli archi adiacenti vengono immessi uno ad uno in una pila ordinata a meno che questo non renda la lunghezza della pila più grande della soglia calcolata. Gli archi che saranno nella pila alla fine dell'iterazione saranno quelli mantenuti dal nodo.

Si può scegliere di mantenere un arco se è mantenuto o da un solo nodo (or) o da entrambi i nodi i nodi che collega (and). Gli archi mantenuti rappresentano le possibili corrispondenze.

A sinistra, il grafo di meta-blocking derivato dall'esempio precedente. Il peso degli archi corrisponde al numero di blocchi in cui i due profili collegati compaiono insieme. A destra, il grafo ottenuto dopo la fase di pruning. Da esso si ottiene si ottengono due blocchi: $b_1 = \{p_1, \{p_3, p_4\}\}$ e $b_2 = \{p_2, \{p_3, p_4\}\}$.

È possibile utilizzare SparkER per casi reali e per effettuare dei test. Per i test, oltre al dataset è necessario dare in input anche un file chiamato "groundtruth" in cui sono presenti tutte le associazioni all'interno del dataset. Dalle associazioni presenti nel groundtruth e dalla raccolta di blocchi ottenuta si raccolgono due parametri per la valutazione dell'algoritmo:

- "Pair Completeness" (PC), definito come il rapporto tra il numero di coppie di profili mantenute nel meta-blocking e il numero di profili corrispondenze nel dataset che viene data in input. Più è grande questo valore, maggiore è l'efficacia;
- "Pair Quality" (PQ), definito come il rapporto tra il numero di coppie di profili mantenute nel metablocking e il numero di corrispondenze trovate. Più è grande questo valore, maggiore è l'efficienza (diminuisce il numero di confronti inutili eseguiti).

1.3 INPUT E OUTPUT

SparkER per tutte le sue operazioni ha bisogno di una serie di parametri che possono essere suddivisi in cinque sezioni:

1)Parametri di Spark, sono parametri per la configurazione di Apache SparK, il framework usato per implementare SparkER. Sono quattro parametri:

- "defaultParallelism", livello di parallelismo;

- "executorMemory", memoria dell'esecutore;
- "stackMemory", quantità di memoria da dare allo stack;
- "sparkLocalDir", percorso assoluto della directory di lavoro di Spark.

2)Parametri per l'ambiente di lavoro, sono tre parametri:
- "runningPath." Stringa che contiene il percorso assoluto della cartella di lavoro, in questa cartella viene creato un log, un file di registrazione sequenziale e cronologica delle operazioni svolte. In esso vengono scritti durante l'esecuzione i tempi e i risultati di ogni operazione;
- "logPath". Stringa che contiene il percorso assoluto della cartella in cui viene spostato il file di log alla fine del processo;
- "id". Valore intero che determina il nome del file di log, ovvero "id.txt" (es, "1.txt");

3)Parametri per la definizione del dataset, sono parametri che riguardano le caratteristiche del dataset o raccolta di entità su chi eseguire l'ER. Sono otto parametri:
- "datasetType". Stringa che contiene il tipo di dataset (Clean o Dirty); nel caso di tipo Clean, vanno dati in input due dataset;
- "datasetRealProfileID". Stringa nome del campo che funge da identificatore univoco per i profili di entità all'interno della raccolta di entità;
- "dataset1Path". Stringa che, nel caso di tipo Clean, contiene il percorso assoluto di uno dei due dataset; nel caso di tipo Dirty, contiene il percorso assoluto dell' unico dataset;
- "dataset2Path". Va inserito solo in caso di tipo Clean ed è una stringa che il percorso assoluto del secondo dataset;
- "groundtruthPath": percorso assoluto del groundtruth;
- "dataset1Type": formato del file del primo dataset (o dell'unico nel caso di tipo Dirty). I valori accettati sono json, csv, serialized;
- "dataset2Type": va inserito solo di dataset di tipo Clean e contiene il formato del file del secondo dataset. I valori accettati sono json, csv, serialized;
- "groundtruthType": contiene il formato del file del groundtruth, i valori accettati sono json, csv, serialized;

3)Parametri per il preprocessing, parametri per le fasi di blocking, block purgin e block filtering. Sono tre parametri:
- "blockingType". Ha due possibili valori; se viene scelto "inferSchema", viene eseguito il clustering degli attributi prima del blocking; se viene scelto "schemaLess" il clustering non viene fatto. Se si sceglie la prima opzione vanno definiti anche altri cinque parametri per il clustering;
- "hashNum". numero di hash che vengono utilizzati nel processo di clustering degli attributi. È un valore intero, un valore tipico è 128. Va definito solo se si sceglie inferSchema come tipo di blocking;
- "clusterThreshold": soglia per il clustering, è un valore double compreso tra 0 e 1. Un valore tipico è 0.3. Va definito solo se si sceglie inferSchema come tipo di blocking;
- "useEntropy": valore booleano che indica se utilizzare o meno l'entropia dei vari cluster per migliorare il pruning successivamente. Va definito solo se si sceglie inferSchema come tipo di

blocking;

- "clusterSeparateAttributes": valore booleano, indica se tenere solo cluster che sono composti da attributi provenienti da entrambi i dataset. Se ad esempio, un cluster contiene solo attributi provenienti da un singolo dataset viene eliminato. Va definito solo se si sceglie inferSchema come tipo di blocking e se il dataset scelto è di tipo Clean;
- "clusterMaxFactor ": è un valore double compreso tra 0 e 1, quando genera i cluster un attributo può finire insieme a tanti altri. Viene calcolata una similarità tra i vari attributi finiti nello stesso cluster, e si mantengono solo quelli che hanno una similarità maggiore o uguali alla similarità maggiore moltiplicata per questo parametro. Quindi più è basso, più ne tiene. Va definito solo se si sceglie inferSchema come tipo di blocking;
- "purgingRatio", valore che definisce il limite superiore per l'operazione di block purging;
- "filteringRatio, valore per il block filtering.

4)Parametri per il meta-blocking. Sono cinque parametri:

- "pruningSchema", tipo di schema di pruning. I valori accettati sono cnp e wnp;
- "thresholdType", indica come viene calcolata la soglia locale di ogni nodo durante la fase di pruning, se si sceglie WNP come schema di pruning (avg o maxdiv2);
- "pruningType", tipologia di pruning, valori accettati and o or;
- "weightTypes", metodi pesatura per gli archi. È un array, può essere indicato più di un metodo. I valori accettati sono: cbs, js, chiSquare, arcs, ecbs, ejs. Per ogni metodo di pesatura viene fatto il meta-blocking e nella cartella dove viene spostato il log viene anche immesso un file con le corrispondenze trovate con quel metodo di pesatura con nome "id+nomeMetodo+.txt" (es, "1cbs.txt");
- "chi2Divider": nel caso in cui tra i metodi di pesatura ci sia il chiSquare, questo parametro viene utilizzato come divisore nel calcolo del peso di un archi. È un valore double che dev'essere maggiore di 0.

2 SPECIFICA DEI REQUISITI

2.1 IDEA DELL'APPLICAZIONE

La necessità di un'applicazione web per collegarsi con SparkER nasce dalla mancanza di un'interfaccia grafica per questo framework. Senza di essa, infatti, risulta poco agevole configurare SparkER, visualizzare e confrontare i risultati di un task. Per lanciare un task, infatti, esso deve ricevere una lunga serie di parametri di input. Inoltre i risultati di un task sono riportati un file di log, un file di monitoraggio che contiene la successione cronologica delle operazioni eseguite e alcuni dettagli riguardo ad esse come il tempo di esecuzione per ogni fase, il numero di profili trovati, il numero di blocchi generati e i risultati generali del meta-blocking (PC, PQ), ecc; in questa situazione non è possibile visionare e confrontare comodamente i risultati dei task.
Lo scopo dell'applicazione è quello di rendere più agevole l'uso di SparkER: l'utente deve poter configurare SparkER per il lancio di un task, monitorare un task durante l'esecuzione e visualizzare i risultati in maniera molto più comoda e intuitiva che permetta anche il confronto immediato tra due o più task.

2.2 REQUISITI FUNZIONALI

I requisiti funzionali descrivono le funzionalità ed i servizi forniti dal sistema (cosa deve essere fatto). Di seguito sono riportate nel dettaglio le macrofunzionalità richieste.

2.2.1 Impostazioni generali

Una pagina deve essere dedicata alle impostazioni per il funzionamento di SparkER, quelle che valgono indipendentemente dalla configurazione dei task lanciati o dai dataset usati. Si tratta dei parametri di funzionamento di Apache Spark e delle cartelle di salvataggio dei log durante la fase di esecuzione e alla fine dell'esecuzione.

2.2.2 Impostazioni dei dataset

Deve essere presente una pagina per poter gestire i dataset su cui eseguire i task di ER, ovvero aggiungerli, modificarli e rimuoverli. Con ciò non si intende la possibilità di fare un upload, perchè i dataset sono spesso di grandi dimensioni, quindi devono essere già presenti sul server. Dalla parte client, per ogni dataset devono semplicemente essere definiti i parametri riguardanti il dataset che SparkER deve ricevere in input quando si intende eseguire un task su quel determinato dataset: il tipo di dataset, il percorso assoluto e il formato. Queste ultime due caratteristiche vanno specificate anche per l'eventuale groundtruth affiancato al dataset.

2.2.3 Configurazione e lancio di un nuovo task

In una pagina dovrà essere possibile lanciare un nuovo task scegliendo il dataset tra quelli disponibili e specificando i parametri per il preprocessing e quelli per il meta-blocking.

2.2.4 Monitoraggio di un task

Deve essere presente una pagina dedicata al monitoraggio dei task in esecuzione in cui sia possibile vedere nel dettaglio la fase in cui essi si trovano (blocking, purging o meta-blocking) e il tempo impiegato per ogni fase.

2.2.5 Visualizzazione e confronto dei task terminati

Una pagina deve essere dedicata alla visione della configurazione e dei risultati di un task, in particolare i tempi e i risultati specifici di ogni fase (numero di blocchi generati, blocchi rimanenti dopo la fase di purging, ecc). Si richiede la possibilità di visualizzare nel dettaglio quanti cluster di attributi sono stati generati durante il clustering (nel caso sia stato scelto il Loose Schema Blocking) e quali attributi contengono. Inoltre i risultati del meta-blocking come PC, PQ e tempo di esecuzione, devono essere rappresentati graficamente. Tutti i risultati devono poter essere confrontati con quelli di un'altro task. Infine, per ogni meta-blocking eseguito in un task, deve essere possibile visualizzare le corrispondenze trovate tra i profili di entità.

2.2.6 Invio dei risultati a determinati utenti di Telegram

Alla fine di un task l'applicazione deve inviare automaticamente una notifica a telegram con i risultati a determinati utenti, i cui numeri di telefono devono poter essere scelti dall'utente tramite l'interfaccia.

2.3 REQUISITI NON FUNZIONALI

I requisiti non funzionali non sono collegati direttamente con le funzioni implementate dal sistema, ma piuttosto alle modalità operative, di gestione. Deniscono vincoli sullo sviluppo del sistema.

2.3.1 Linguaggi utilizzati

Per la parte client è necessario l'utilizzo di HTML5, JavaScript e CSS, mentre per la parte server è stato espressamente richiesto il linguaggio Python. Sulla macchina in cui risiederà l'applicazione dovrà essere installata la versione 3.0 (o superiore) dell'interprete classico Python poichè le versioni precedenti non sono compatibili con l'attuale standard del linguaggio.

2.3.2 Responsive web design

Nello sviluppo dell'applicazione è richiesto l'utilizzo della tecnica del responsive web design, ovvero le pagine web devono poter adattare automaticamente il layout a diversi dispositivi come pc, tablet e smartphone.

2.3.3 Database

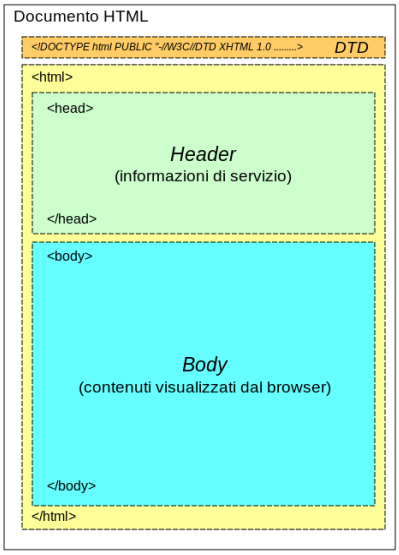
L'applicazione deve essere in gado di funzionare sul computer ad alte prestazioni PICO di CINECA, sul quale non è possibile installare alcun tipo di DBMS. Per questo è stato espressamente richiesto che le impostazioni generali, delle notifiche, i parametri dei dataset e le configurazioni dei dataset vengano salvati tramite dei JSON.

3 TECNOLOGIE USATE

3.1 HTML

L'HyperText Markup Language (letteralmente "Linguaggio a marcatori per ipertesti") è un linguaggio di markup (di 'contrassegno' o 'di marcatura'). Inizialmente permetteva sia la formattazione che l'impaginazione di documenti ipertestuali, mentre oggi è usato principalmente per disaccoppiare la struttura logica di una pagina web (definita appunto dal markup) e la sua rappresentazione, la quale è gestita tramite gli stili CSS.
L'HTML è un linguaggio di pubblico dominio, la cui sintassi è stabilita dal World Wide Web Consortium (W3C).

Ad oggi, l'HTML è un linguaggio di formattazione che descrive le modalità di impaginazione o visualizzazione grafica (layout) del contenuto, testuale e non, di una pagina web attraverso degli appositi marcatori, detti **tag** ('etichette'), che hanno la caratteristica di essere inclusi tra parentesi angolari (ad es, <head>, <body>, <img>). Il nome del tag esplicita la tipologia di elemento, ovvero se si tratta di un titolo, di un paragrafo, un riferimento ipertestuale, un campo di input, ecc. All'interno del tag possono essere inseriti degli attributi che delineano altre caratteristiche dell'elemento quali come la funzione, il colore, le dimensioni, la posizione relativa all'interno della pagina e i legami con il codice JavaScript.



Struttura generica di un documento HTML

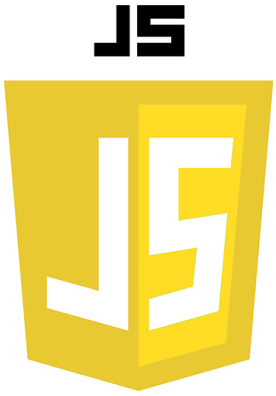
L'HTML supporta anche l'inserimento di script e oggetti esterni quali immagini o filmati, ma non possiede i costrutti propri della programmazione quali definizione di variabili, strutture dati, funzioni o strutture di controllo che possano costituire programmi. Non è pertanto considerabile come un linguaggio di programmazione. Quando un documento ipertestuale scritto in HTML è memorizzato in un file la sua estensione è tipicamente .html o .htm.

I documenti HTML vengono immagazzinati sui dischi rigidi di macchine elaboratrici (computer-server) costantemente collegate e connesse alla rete Internet. Su queste macchine è installato un software specifico (web server) che si occupa di produrre e inviare i documenti ai browser degli utenti che ne fanno richiesta usando il protocollo HTTP per il trasferimento dati. Quando ciò avviene la pagina richiesta viene elaborata dal browser dell'utente. Il risultato dell'elaborazione rappresenta la visualizzazione sullo schermo della pagina.
L'ultima versione del linguaggio, HTML5, rilasciata nell'ottobre del 2014, introduce novità che rinforzano il disaccoppiamento tra struttura e caratteristiche di resa e aggiungono funzionalità dinamiche per cui prima era necessario codice javascript.

Nella realizzazione dell'applicazione per SparkER questo linguaggio è stato utilizzato per la creazione delle pagine e viste e le potenzialità dell'ultima versione sono state ampiamente

sfruttate soprattutto perché indispensabili per l'utilizzo del framework AngularJS.

3.2 JAVASCRIPT



JavaScript è un linguaggio di scripting orientato agli oggetti e agli eventi, comunemente utilizzato nella programmazione Web lato client per la creazione, in siti e applicazioni web, di effetti dinamici interattivi tramite funzioni di script, le quali sono invocate da eventi innescati a loro volta in vari modi dall'utente (mouse, tastiera, caricamento della pagina ecc.). Tali funzioni di script, utilizzati dunque nella logica di presentazione, possono essere opportunamente inserite in file HTML, in pagine JSP o in appositi file separati con estensione .js.
Nonostante il nome, non c'è una vera relazione tra Java e JavaScript in quanto le loro somiglianze sono soprattutto nella sintassi, ma non per quanto riguarda le loro semantiche le quali sono piuttosto diverse, e in particolare i loro object model non hanno relazione e sono ampiamente incompatibili.

Le caratteristiche principali di JavaScript sono:

- l'essere un linguaggio interpretato (per il JavaScript lato client l'interprete è incluso nel browser che si sta utilizzando);
- la sintassi è relativamente simile a quella del C, del C++ e del Java;
- definisce le funzionalità tipiche dei linguaggi di programmazione ad alto livello (strutture di controllo, cicli, ecc.) e consente l'utilizzo del paradigma object oriented;
- è un linguaggio a tipizzazione debole e dinamica;
- è un linguaggio debolmente orientato agli oggetti;
- il codice viene eseguito direttamente sul client e non sul server. Il vantaggio di questo approccio è che il web server non viene mai sovraccaricato a causa delle richieste dei client, ma il tempo per lo scaricamento può diventare abbastanza lungo.

Le interfacce che consentono a JavaScript di rapportarsi con un browser sono chiamate DOM (*Document Object Model* in italiano *Modello a Oggetti del Documento*). Molti siti web usano la tecnologia JavaScript lato client per creare potenti applicazioni web dinamiche. Un uso principale del JavaScript in ambito Web è la scrittura di piccole funzioni integrate nelle pagine HTML che interagiscono con il DOM del browser per compiere determinate azioni che, normalmente, non sarebbero possibili con il solo HTML statico: controllare i valori nei campi di input, nascondere o visualizzare determinati elementi, creare animazioni, ecc.

All'interno dell'applicazione per SparkER, JavaScript è stato ampiamente usato secondo l'architettura impostata da AngularJS.

3.3 CSS

Il CSS (Cascading Style Sheets, ovvero fogli di stile a cascata), è un linguaggio usato per definire la formattazione di documenti HTML, XHTML e XML. Si occupa quindi di tutte quelle caratteristiche che

vanno ad interessare la presentazione della pagina, ossia come vengono visualizzati gli elementi definiti nel documento HTML. Comprende, tra le varie cose, la definizione delle caratteristiche dei font, colori, bordi, margini, posizione relativa di un elemento, e molto altro.

L'introduzione del CSS si è resa necessaria per separare i contenuti delle pagine HTML dalla loro formattazione e permettere una programmazione più chiara e facile da utilizzare, sia per gli autori delle pagine stesse sia per gli utenti, garantendo contemporaneamente anche il riutilizzo di codice ed una sua più facile manutenzione. Un altro vantaggio è quello della definizione delle caratteristiche per dei gruppi di elementi, aggregati attraverso l'uso di selettori quali i selettori di tipo, le classi o un identificativo.

L'ultima versione, la 3.0, prevede che le specifiche dei CSS siano divise in moduli con diversi stati di avanzamento e stabilità e introduce nuove possibilità per quanto riguarda bordi, sfondi, testi e layout. Le maggiori novità della versione 3.0, però, sono l'introduzione di effeti dinamici (transizioni, trasformazioni e animazioni) e delle "media queries"; queste rinforzano il meccanismo dei precedenti "media types" (CSS2) di servire fogli di stile ad hoc a seconda del dispositivo di visualizzazione.

L'inserimento di codice CSS nelle pagine web può essere effettuato in tre modi:

- Inserendo nel tag <head> della pagina in codice HTML un collegamento ad un foglio di stile esterno, cioè un file di tipo .css, tramite il tag link o tramite la direttiva import. Se vengono inseriti più fogli e le loro regole sono in contrasto, prevarranno quelle del foglio corrispondente all'ultimo collegamento inserito (erediarietà a cascata);
- Inserendo, sempre all'interno del tag <head> tra gli specifici tag <style> e </style> le dichiarazioni css. Le regole scritte in questi spazi prevalgono su quelle scritte nei fogli esterni collegati alla pagina;
- In linea, all'interno degli elementi <tag> e </tag> tramite la sintassi style="dichiarazioni CSS". Le regole così scritte prevalgono su quelle inserite nelle due modalità precedenti.

Un foglio di stile CSS è sintatticamente strutturato come una sequenza di regole, che sono coppie costituite da un selettore e un blocco di dichiarazioni, racchiuso tra parentesi graffe. Un selettore è un predicato che individua certi elementi del documento HTML; una dichiarazione, separata con un punto e virgola dalle altre, è a sua volta costituita da una proprietà, ovvero un tratto di stile e un valore da assegnare a quest'ultimo separati dai due punti.

Nell'applicazione per SparkER, per la formattazione delle pagine web sono stati creati due fogli di stile per le caratteristiche di stile particolari dell'applicazione, ma lo stile è stato perlopiù determinato dagli elementi presi da Bootstrap, che hanno già caratteri estetici predefiniti.

3.4 BOOTSTRAP

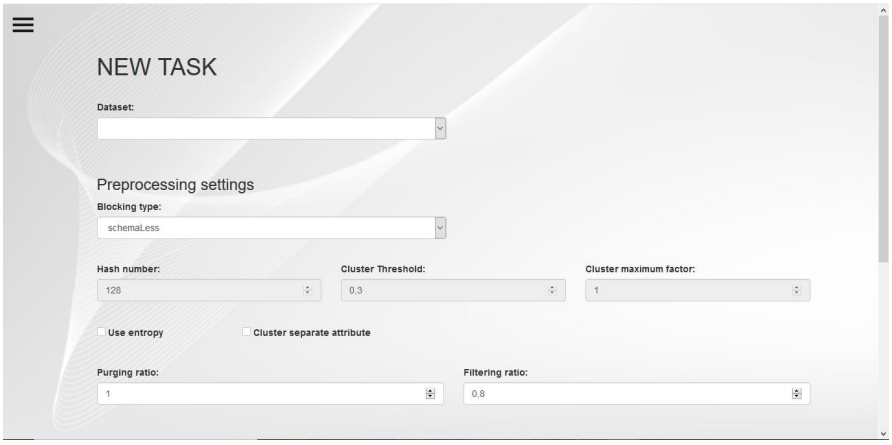
Bootstrap è un framework per siti e applicazioni web. Esso contiene modelli di progettazione basati su HTML e CSS, sia per la tipografia, che per le varie componenti dell'interfaccia, come moduli, pulsanti e navigazione, così come alcune estensioni opzionali di JavaScript, contenenti soprattutto gestioni particolari

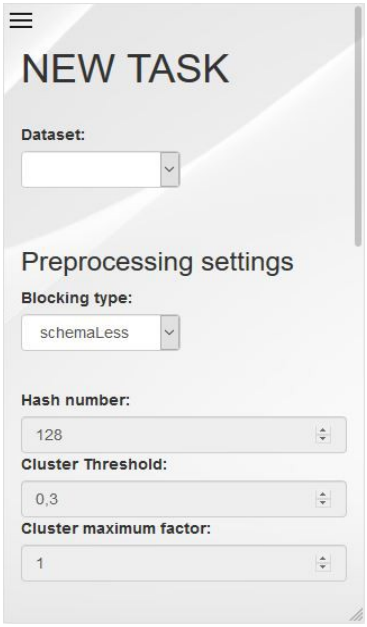
di effetti, quali animazioni e transizioni.

Attualmente, le principali caratteristiche di questo framework sono le seguenti:

- Supporta il responsive web design; dalla versione 3.0, di default, il layout delle pagine web basate su Bootstrap si regola dinamicamente, tenendo conto delle caratteristiche del dispositivo utilizzato, sia esso desktop, tablet o smartphone. Per permettere ciò, il framework fa un ampio uso delle media queries del CSS;
- Si presenta nel suo look and feel predefinito con una veste estetica più semplificata rispetto alle versioni precedenti, adattandosi in parte alla tendenza del flat design;
- Contiene un ampio pannello di personalizzazione del framework. Bootstrap è solo un tool per gestire al meglio la fase di avvio di un progetto, un modo per poter contare su una serie di componenti riusabili e personalizzabili, comunque adattabili in termini stilistici ed estetici alle richieste del progetto e alla creatività di chi concepisce il sito.

Nel caso dell'applicazione per SparkER, si richiedeva che l'applicazione fosse visualizzabile anche da dispositivi mobili, perciò è stata necessaria l'adozione del web responsive design. Bootstrap è stato lo strumento suggerito per adempiere a tale necessità. Per quanto riguarda lo stile grafico, poche sono le personalizzazioni introdotte rispetto allo stile predefinito del framework.





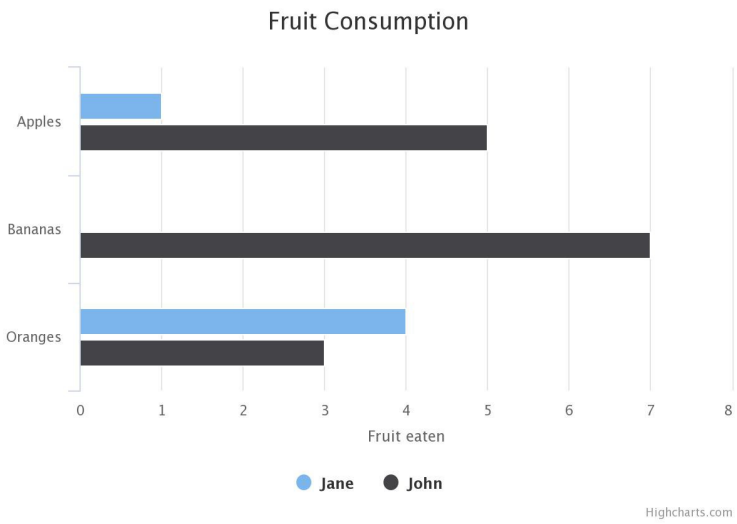
A sinistra, una delle schermate dell'applicazione visualizzata da un pc (`1366x768`). A fianco, la medesima schermata adattata automaticamente alle dimensioni di uno smartphone (`320x568`)

3.5 HIGHCHARTS

Highcharts è una libreria JavaScript open source per la realizzazione di diverse tipologie di grafici, basato sulle tecnologie native dei browser e su SVG, una tecnologia per la manipolazione di forme geometriche e immagini digitali. È stata realizzata da Highsoft, società con sede in Norvegia. Si tratta di un libreria flessibile, dinamica e facile da usare con la quale è possibile realizzare diversi tipi di grafico. Sul sito ufficiale dei Highcharts

(www.highcharts.com) sono già presenti numerose demo per la realizzazione dei grafici più comunemente usati, quali i grafici a barre o a colonne, quelli a torta, i grafici a linee, quelli ad area, quelli radiali , ecc.

```
Highcharts.chart('container', {
    chart: {
        type: 'bar'
    },
    title: {
        text: 'Fruit Consumption'
    },
    xAxis: {
        categories: ['Apples', 'Bananas', 'Oranges']
    },
    yAxis: {
        title: {
            text: 'Fruit eaten'
        }
    },
    series: [{
            name: 'Jane',
            data: [1, 0, 4]
        },
        {
            name: 'Jane',
            data: [5, 7, 3]
        }]
});
```



Un esempio di grafico semplice realizzato con Highcharts. A sinistra il codice javascript, a destra il risultato. Tramite il semplice inserimento o modifica delle proprietà dell'oggetto passato al metodo "chart()", si può definire il grafico. La libreria highcharts.js, opportunamente inclusa nel file HTML e richiamata tramite l'oggetto *Highcharts*, in base alle proprietà specificate e ai loro valori, elaborerà il grafico e lo visualizzera nell'elemento del DOM identificato come *container*.

Un test eseguito da SparkER produce molti risultati, tra i quali vi sono molti dati numerici. Era richiesto che i dati più cruciali tra questi fossero analizzabili e confrontabili con quelli di altri test per via grafica. Perciò è stato consigliato l'utilizzo di Highcharts.

3.6 ANGULARJS

AngularJS (o semplicemente Angular o Angular.js) è un framework web open source principalmente sviluppato da Google. Esso per affrontare le molte difficoltà incontrate nello sviluppo di "Single-Page Applications".

AngularJS si basa sulla convinzione che la programmazione dichiarativa debba essere utilizzata per creare interfacce utente e connettere componenti software, mentre la programmazione imperativa è più adatta alla definizione della logica di business di un'applicazione. Esso, infatti, potenzia l'approccio dichiarativo di HTML nella definizione dell'interfaccia grafica, rendendolo capace di manipolare dinamicamente gli elementi di un documento HTML. Come risultato si elimina la necessità di manipolare esplicitamente il DOM attraverso JavaScript, cosa che facilita la testabilità e la performace di un'applicazione web.

3.6.1 Componenti di AngularJS

Nella realizzazione di un'applicazione, AngularJS propone una strutturazione modulare composta da componenti che hanno ciascuno una ben precisa funzione, seguendo il principio della separazione delle competenze. I principali componenti sono:

- **View**. Rappresenta quello che l'utente vede, l'interfaccia grafica generata a partire da un template HTML elaborato da Angular;

- **Controller**. È un oggetto JavaScript che espone dati e funzionalità ad una view.
- **Filtro**. È una funzione che formatta il valore di un'espressione per la visualizzazione su una view.
- **Direttiva**. È questa la componente che estende la logica dell'HTML con tag ed attributi personalizzati (questi contengono di solito il prefisso *ng-*) ;
- **Servizio**. È un oggetto che fornisce funzionalità indipendenti dall'interfaccia grafica, come ad esempio l'accesso al server via HTTP;
- **Modulo**. È un contenitore di componenti, indipendente dalla loro natura e dalla loro collocazione fisica;

AngularJS supporta il pattern della Dependency Injection, il quale prevede che, quando un componente viene dichiarato, vengano dichiarate anche le dipendenze da componenti già esistenti.

Nell'applicazione per SparkER, molte funzioni normalmente affidate a JavaScript sono state delegate a moduli preesistenti trovati in rete, come la gestione degli effetti dinamici di Bootstrap, o l'inserimento dei grafici di Highcharts.

3.6.2 MVC e data binding

La separazione delle competenze, in AngularJS, avviene secondo il pattern MVC (Model-View-Controller). Tramite la direttiva *ng-app* viene indicato ad Angular quale elemento della pagina prendere in considerazione per poterlo interpretare come interfaccia grafica della nostra applicazione. Questa interfaccia può essere collegata a un modulo che contenga tutti i componenti della nostra applicazione, con il metodo *module()*. Nel modulo è possibile poi, tramite *controller(),* definire un controller con una stringa e una funzione di costruzione. Con la direttiva *ng-controller*, esso è associabile a una parte dell'interfaccia grafica contrassegnata da *ng-app*. Questa parte di intefaccia diventerà, quindi, a tutti gli effetti la view manipolata dal controller.

```
<body ng-app="myApp">
    <div ng-controller="userController">
        <p>Nome: <input type="text" ng-model="nome"></p>
        <p>Cognome: <input type="text" ng-model="cognome"></p>
        <div>Buongiorno {{nome}} {{cognome}}!</div>
    </div>

    <script type="text/javascript">
        app = angular.module("myApp", []);
        app.controller("userController", function($scope) {
            $scope.nome = "Mario";
            $scope.cognome = "Rossi";
        });
    </script>
</body>
```

Nome: Mario

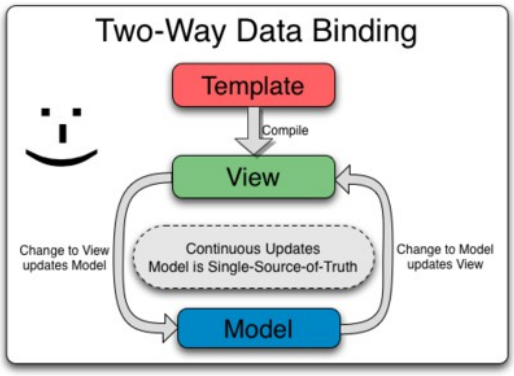
Cognome: Rossi

Buongiorno Mario Rossi!

A sinistra, la definizione del controller e di una vista, a destra l'output corrispondente.

Il modello dei dati è invece definito nella funzione di costruzione tramite lo $scope, parametro passato al controller dal framework e condiviso con la view. Tutte le proprietà di questo oggetto, infatti, sono accessibili alla view attraverso le direttive (es. *ng-model, ng-repeat, ng-init, ng-click, ecc*). Questa sincronizzazione automatica dei dati tra il modello e la view è detta "data binding".

La maggior parte dei sistemi di template supporta il data binding in una sola direzione, tipicamente dal modello dei dati verso la view. Questo vuol dire che i dati del modello vengono combinati con il template HTML per generare la view visibile all'utente. Inoltre, di solito il meccanismo non è automatico, quindi, per sincronizzare view e modello occorre in genere scrivere del codice che lo faccia. Il data binding di AngularJS invece è bidirezionale ("two-way data binding") e automatico, cioè ogni modifica al modello dei dati si riflette automaticamente sulla view e ogni modifica alla view viene riportata sul modello dei dati. Tutto questo con il semplice uso dello scope e delle direttive. Una direttiva lega l'elemento della vista in cui è presente come tag a un dato del modello. Se questo dato viene modificato durante l'uso dell'applicazione, l'elemento viene modificato immediatamente di conseguenza e, viceversa. Nell'esempio sopra, i valori dei campi di input vengono legati ai valori delle proprietà *nome* e *cognome* dello scope dalla direttiva *ng-model*. Se il valore di un campo di input viene modificato, tale modifica ricade sulla proprietà a cui l'elemento è legato.



Schema del Two-way data binding

3.6.3 Validazione di un form

Un form è un termine utilizzato per riferirsi all'interfaccia di un'applicazione che consente all'utente client di inserire e inviare al web server uno o più dati liberamente digitati dallo stesso. Di solito i form vengono utilizzati per inviare dati ad un database server oppure per inviare e-mail. Tutti i dati di un form, una volta inviati dal web browser del client al web server, devono essere trattati o elaborati lato server.

HTML5 presenta un serie di attributi per il controllo della correttezza dei dati inseriti come *required*, *maxlength*, *pattern*, ecc. Ad essi corrispondono delle direttive Angular come *ng-required*, *ng-maxlength*, *ng-pattern*, etc. Il valore assegnato a una direttiva viene automaticamente assegnato all'attributo corrispondente. La duplicazione sembrerebbe essere inutile ma le direttive Angular sono più flessibili, perchè sono appunto visibili dall'ambiente Angular tramite il two-way data binding.

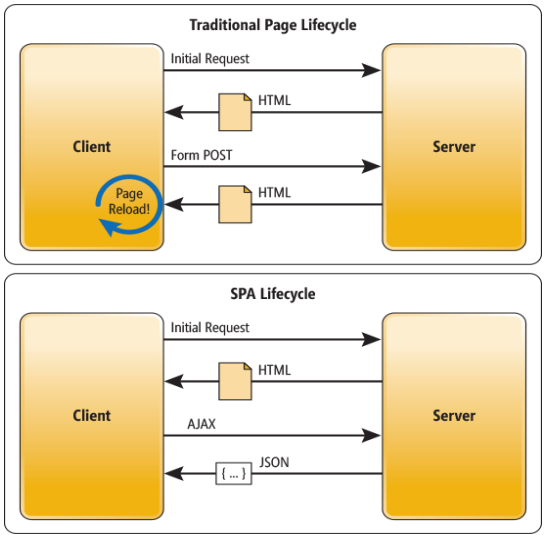
```
<form name="myForm" ng-submit="aggiungiUtente(utente)" novalidate>

    <label>Name: <input type="text" ng-model="utente.name" ng-required="true"/></label>
    <label>Cittadinanza italiana: <input type="checkbox"
                                         ng-model="utente.cittadinanzaItaliana"></label>

    <label>Cittadinanza: </label><input type="text" ng-model="utente.cittadinanza"
                                        ng-required="!utente.cittadinanzaItaliana"
                                        ng-disabled="utente.cittadinanzaItaliana"></label>

      <input type="submit" ng-disabled="myForm.$invalid" />

</form>
```

Esempio di form con direttive Angular

Nell'esempio, l'attributo *novalidate* impedisce al browser di ignorare i controlli di validazione di AngularJS. Nel campo di input 'Cittadinanza', si possono notare le direttiva *ng-disabled* e *ng-required*. Questi attributi assumono valore *true* o *false* in base al valore all'esistenza della variabile *utente.cittadinanzaItaliana,* la quale assume valore dal checkbox 'Cittadinanza Italiana'. In altre parole se *utente.cittadinanzaItaliana* sarà "true", allora il campo di testo verrà disabilitato e non sarà richiesto, altrimenti non sarà disabilitato e sarà obbligatorio inserire un valore. Il pulsante del form sarà abilitato solo se gli attributi richiesti sono stati inseriti e validi (*ng-disabled = myForm. $invalid*).
Le direttive AngularJS permettono di far variare le validazioni dei campi di input in base a determinate situazioni. Questa possibilità è stata ampiamente utilizzata nell'applicazione per SparkER.

3.6.4 Single-page application

Una Single-page application (SPA) è un'applicazione Web che interagisce con l'utente riscrivendo dinamicamente la pagina corrente anziché caricare intere nuove pagine da un server. Questo approccio evita l'interruzione dell'esperienza utente tra pagine successive.



Funzionamento di un'applicazione web tradizionale (sopra) e di un SPA (sotto)

Lo scenario tipico è quello di una sola pagina HTML la quale scarica tutti i file JavaScript e CSS necessari all'applicazione. Essa contiene un tag (tipicamente un div) in cui frammenti HTML vengono immessi dinamicamente da JavaScript dopo essere stati scaricati dal server tramite AJAX. Si ha, quindi, l'illusione di passare da una pagina a un'altra, ma in realtà, si ha l'alternanza, sulla pagina principale, di frammenti di pagina chiamati 'viste', a ciascuna delle quali è comunque mappato un URL "fittizio" (routing) per la navigazione.
AJAX prevede anche di poter scaricare in modo asincrono dei dati in formato JSON (JavaScript Objects Notation), i quali vengono caricati in background senza interferire con l'operatività della pagina corrente. Se quest'ultima deve essere poi modificata in base ai dati presi dal server, ciò potrà avvenire direttamente sul browser, senza dover passare per un ricaricamento della pagina intera. L'eliminazione dei caricamenti di pagine permette un'esperienza simile a quella delle applicazioni desktop.

JSON è un formato di oggetto usato in AJAX come alternativa a XML/XSLT. La sua semplicità è dovuta alla facilità con cui l'interprete JavaScript sia in grado di effettuare il parsing. La maggior parte dei linguaggi di programmazione possiede un typesystem molto simile a quello definito da JSON per cui sono nati molti progetti che permettono l'utilizzo di JSON con altri linguaggi, quali Java, Python, Javascript, PHP, C, C# e molti altri. I tipi di dato supportati da JSON sono i booleani,

numeri interi, reali e in virgola mobile, stringhe, array e array associativi. La sintassi è basata su due strutture:

- Un insieme di coppie *nome-valore*. In diversi linguaggi, questo è realizzato come un oggetto, un record, uno struct, un dizionario, una tabella hash, un elenco di chiavi o un array associativo.
- Un elenco ordinato di valori. Nella maggior parte dei linguaggi questo si realizza con un array, un vettore, un elenco o una sequenza.

```
{
    "type": "menu",
    "value": "File",
    "items": [
            {"value": "New", "action": "CreateNewDoc"},
            {"value": "Open", "action": "OpenDoc"},
            {"value": "Close", "action": "CloseDoc"}
    ]
}
```

Esempio di oggetto JSON

Tra i tanti moduli già esistenti di AngularJS ve ne è uno per la realizzazione delle SPA, ovvero *ngRoute.* Esso contiene il servizio per il routing che permette anche di associare un controller alle viste (*$routeProvider*). Vi è poi la direttiva *ng-view* per definire il contenitore delle viste sulla pagina principale. Una volta impostati questi due elementi, la navigazione tra le viste sarà già funzionante; normalmente, invece, sarebbe necessario del codice JavaScript per catturare la richiesta di navigazione verso una vista, prendere la vista corrispondente e inserirla nel contenitore.
Inoltre, tutti i template HTML e i controller associati ad essi vengono portati sul browser all'avvio dell'applicazione e vengono mantenuti durante tutta la fase di uso dell'applicazione. Nel pacchetto base di AngularJS è, inoltre, presente il servizio *$http* per le richieste AJAX.

In conclusione, AngularJS è stato usato nell'applicazione per SparkER per i seguenti motivi: esso offre la possibilità di realizzare una SPA in cui l'interfaccia grafica è totalmente staccata dal JavaScript, lasciando a quest'ultimo il compito di definire la logica applicativa, in particolare di interfacciarsi con il server, inviando e ricevendo dati; tutto ciò evita la necessità di ricaricamenti di pagine. L'uso di AJAX, inoltre, si concilia perfettamente con la richiesta di utilizzare dei file JSON per mantenere i dati relativi ai test.

3.7 PYTHON

Python è un linguaggio di programmazione ad alto livello ideato dall'informatico olandese Guido Van Rossum all'inizio degli anni '90. È utilizzato soprattutto per lo sviluppo di script, applicazioni distribuite e per effettuare system testing.
Python ha tra i principali obiettivi dinamicità, semplicità e flessibilità. Ha infatti una sintassi semplice che ne facilita l'apprendimento; supporta diversi paradigmi di programmazione (strutturata, a oggetti e funzionale); vanta un ricco assortimento di tipi, funzioni di base e librerie

standard.
Python, visivamente, si presenta in modo facilmente leggibile. Ciò è dovuto soprattutto all'obbligo di usare l'identazione in congiunzione col carattere "due punti" (:) per indicare i blocchi di codice e separarli, cosa che nella maggior parte dei linguaggi è responsabilità del programmatore. Supporta tutti i costrutti classici dei linguaggi di programmazione quali i cicli, i costrutti condizioni (*if…elif…else*) e quelli per la gestione delle eccezioni (*try…except…finally*).
Python è un linguaggio a tipizzazione forte e dinamica. Gli oggetti hanno dei tipi ben definiti, ma le variabili sono semplicemente delle "etichette" associabili a un oggetto di qualsiasi tipo (sono dei puntatori). Le variabili non sono quindi legate al tipo di oggetto a cui puntano e possono, quindi, passare da un tipo a un altro durante il loro ciclo di vita. Tutto ciò è possibile perché il controllo dei tipi, comunque forte, viene eseguito in fase runtime e non al momento della compilazione. A questo si aggiunge un garbage collector per la liberazione automatica della memoria.
Python mette a disposizione un gran numero di tipi base, essenzialmente tipi numerici e contenitori. Caratteristica distintiva è il supporto nativo, oltre che ai classici tipi quali interi, floating point, stringhe, anche a tipi più evoluti quali interi a grandezza arbitraria, numeri complessi, liste, insiemi, dizionari, con delle comode sintassi per la costruzione degli stessi (iteratori, generatori e list comprehension). Non è invece previsto un tipo specifico per i caratteri.
Il sistema dei tipi Python è ben integrato con il sistema delle classi. Anche se i tipi base non sono precisamente classi, una classe può ereditare da essi. In questo modo è possibile estendere stringhe, dizionari e perfino gli interi. Sono inoltre supportati:

- l'ereditarietà singola e multipla;
- l'overloading degli operatori, ovvero la possibilità di definire una famiglia di funzioni o metodi di classe con lo stesso nome ma con un diverso insieme di argomenti;
- il duck typing. La semantica di un oggetto è determinata dall'insieme corrente dei suoi metodi e delle sue proprietà anziché dal suo tipo o da eventuali classi estese.

Python è un linguaggio interpretato. Un interprete si occupa di analizzare il codice sorgente (descritto in file di testo con estensione .py) e, se sintatticamente corretto, di eseguirlo. Questa caratteristica lo rende portabile. Una volta scritto un sorgente, esso può essere interpretato ed eseguito sulla gran parte delle piattaforme attualmente utilizzate. Per una corretta esecuzione è sufficiente, infatti, la presenza della versione corretta dell'interprete.
A differenza degli altri linguaggi, però, vi è una fase preliminare in cui il file .py viene compilato in un formato detto "bytecode". Questo formato è più compatto ed efficiente, e garantisce quindi prestazione elevate. A ciò si aggiunge che nell'interprete classico, CPython (scritto in C), sono integrati funzioni e strutture dati Python direttamente implementati in C per essere ancora più performanti.
Python può essere integrato con altri linguaggi, come, ad esempio, .NET, attraverso IronPython, o Java, attraverso Jython.

Nell'applicazione per SparkER, Python è stato scelto per lo sviluppo della parte server, la quale contiene le funzioni per avviare un test con SparkER e raccogliere i dati. Queste sono le funzioni chiamate dal client via AJAX.

3.7.1 Flask

Flask è un framework web leggero scritto in Python. Si tratta di un "microframework" perché ha un nucleo semplice ma estensibile. Di default, Flask non include uno strato di astrazione per i database, la validazione dei form o qualsiasi altra cosa per cui esistono già librerie esterne in grado di gestirle. Esso, invece, supporta delle estensioni per aggiungere queste funzionalità a un'applicazione come se fossero implementate in Flask stesso. Tutto ciò fa di Flask un framework leggero ma pronto ad rispondere a qualsiasi esigenza senza che sia l'applicazione a doversi adattare al framework. Tra le altre caratteristiche, troviamo che Flask:

- contiene server e debugger per lo sviluppo;
- contiene il supporto integrato per il test d'unità;
- usa Jinja2 come motore di template;
- supporta cookie di sicurezza;
- è basato sullo strumento Werkzeug WSGI (Web Server Gateway Interface), un protocollo di trasmissione che stabilisce e descrive comunicazioni ed interazioni tra server e applicazioni web scritte in Python;
- è basato su Unicode;
- ha una documentazione estensiva;
- compatibilità con Google App Engine, una piattaforma come servizio di cloud computing per lo sviluppo e l'hosting di applicazioni web gestite dai Google Data Center.

Inizialmente, era stato suggerito di utilizzare Apache HTTP come server web e di usare Python solo per scrivere le funzioni dell'applicazione server. Successivamente si è deciso di usare un server web scritto direttamente in Python per maggiore comodità e portabilità. In precedenza a questa scelta si era anche stabilito di non connettere l'applicazione a un database e che la validazione dei dati venisse fatta tutta lato client con AngularJS. Il risultato è stato che Python si dovesse limitare a ricevere le richieste dal client ed elaborare le risposte. Per queste ragioni, Flask è stato preferito a framework più popolari come Django (più adatto a progetti di grandi dimensioni) o Pyramid (più flessibile, ma meno semplice e con tante funzionalità che non sarebbero state usate).

```python
from flask import Flask, render_template

app = Flask(__name__)

@app.route('/')
def index():
    return render_template('index.html')

@app.route('/changeSettings', methods=['POST'])
def changeSettings():
    return "Settings changed"

if __name__ == '__main__' :
    app.run(debug=True)
```

```javascript
$http({
    method: 'POST',
    url: "/changeSettings",
    datatype: "json",
    contentType:"application/json",
    data: datasend

}).then(function successCallback(response) {

    $log.info(response.data);

}, function errorCallback(response) {
    $log.info(response.status);
});
```

A sinistra, un server web scritto con Flask. Tramite il decoratore *app.route()*, si definiscono degli URL a cui associare delle funzioni. Queste funzioni potranno poi essere chiamate via AJAX, come nell'immagine a destra.

3.8 TELEGRAM

Telegram è un servizio di messaggistica istantanea basato su cloud ed erogato senza fini di lucro dalla società *Telegram LLC*. I client ufficiali di Telegram sono software libero, mentre il codice sorgente della parte server invece non è stato rilasciato.

Il servizio prevede le classiche **chat cloud**, ovvero, chat con cifratura *client-server* che fa su che le conversazioni rimangano salvate in maniera cifrata sui server di Telegram. Ciò permette la sincronizzazione istantanea tra più dispositivi usati dallo stesso utente, il quale può accedere alle proprie conversazioni da più dispositivi anche contemporaneamente (smartphone, tablet, pc, ecc).
Vi sono anche le **chat segrete**, con cifratura *end-to-end*. In questo caso non vi è salvataggio sui server e non c'è possibilità di sincronizzazione.
Sono previste le chat di gruppo, divise in gruppi (con capienza massima di duecento membri) e supergruppi (con capienza massima di cinquantamila membri e delle funzionalità in più).

Da giugno 2015 Telegram ha introdotto una piattaforma per permettere, a sviluppatori terzi, di creare i bot. Il bot, in generale, è un programma che accede alla rete attraverso lo stesso tipo di canali utilizzati dagli utenti umani (per esempio che accede alle pagine Web, invia messaggi in una chat, si muove nei videogiochi, e così via).
Su Telegram, i bot sono degli account speciali che non richiedono numeri di telefono e sono gestibili da programmi esterni al servizio di messaggistica. Per creare un bot è necessario interagire con *BotFather*. Al momento della creazione, viene rilasciato un codice identificativo del bot creato, attraverso il quale sarà possibile collegare tale bot a un oggetto del codice sorgente di un'applicazione. Tramite questo oggetto, l'applicazione potrà ricevere le notifiche degli utenti del bot e inviarne a sua volta o come risposta o come parte di un processo in esecuzione sull'applicazione. La comunicazione tra l'applicazione e il bot passa per i server di Telegram, i quali gestiscono tutte le funzionalità di crittografia e comunicazione con l'API Telegram. Tale comunicazione è gestita da una semplice interfaccia, chiamata HTTPS che offre una versione semplificata dell'API di Telegram.

Per molti linguaggi esistono delle librerie esterne per la comunicazione con l'API di Telegram. Telepot è la più popolare per il linguaggio Python. Essa è stata usata nell'applicazione per SparkER.

4 REALIZZAZIONE DELL'APPLICAZIONE

4.1 ARCHITETTURA DEL SISTEMA SOFTWARE

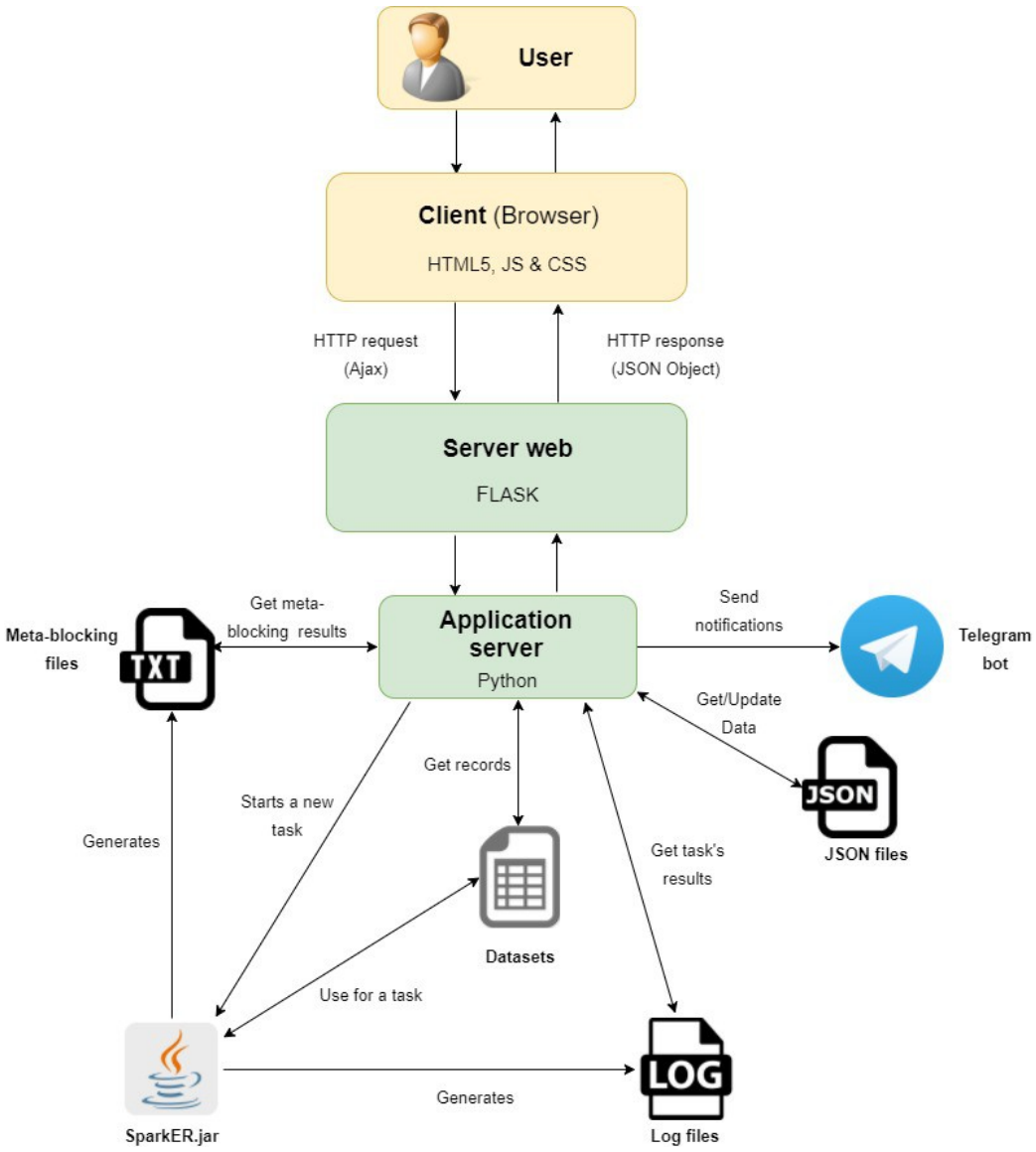


Immagine dell'architettura del sistema software realizzato

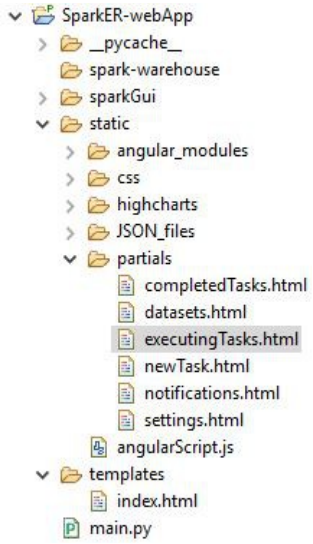
L'architettura dell'applicazione segue il modello generale di un applicazione web, ovvero il modello client-server. Le particolarità stanno nei dettagli implementativi. Gli elementi principali sono:

- **Client**. È il software che costituisce l'interfaccia con cui l'utente interagisce. Esso manda delle richieste al server per accedere ai servizi del server stesso. Nell'applicazione per SparkER, il client è costituito dal browser attraverso cui l'utente accede alle pagine dell'applicazione.
- **Application Server**. È il modulo software che implementa la gestione logica del sistema, assieme a tutte le tecniche di gestione degli accessi, allocazione e rilascio delle risorse, condivisione e sicurezza di dati e risorse. Esso si collega alla rete internet tramite un server

web.

- **Server web**. È un software in esecuzione su un terminale è in grado di gestire le richieste di trasferimento di pagine web di un client, tipicamente un web browser. La comunicazione tra il browser e il server web avviene tramite il protocollo HTTP.
- **Risorse sul server**. Sono le risorse gestite dal server per il funzionamento dell'applicazione.
- **Bot di Telegram**. L'account di telegram a cui il server è collegato tramite la libreria telepot.

4.1.1 Struttura del sito web



Disposizione dei file dell'applicazione secondo la grafica dell'IDE eclipse

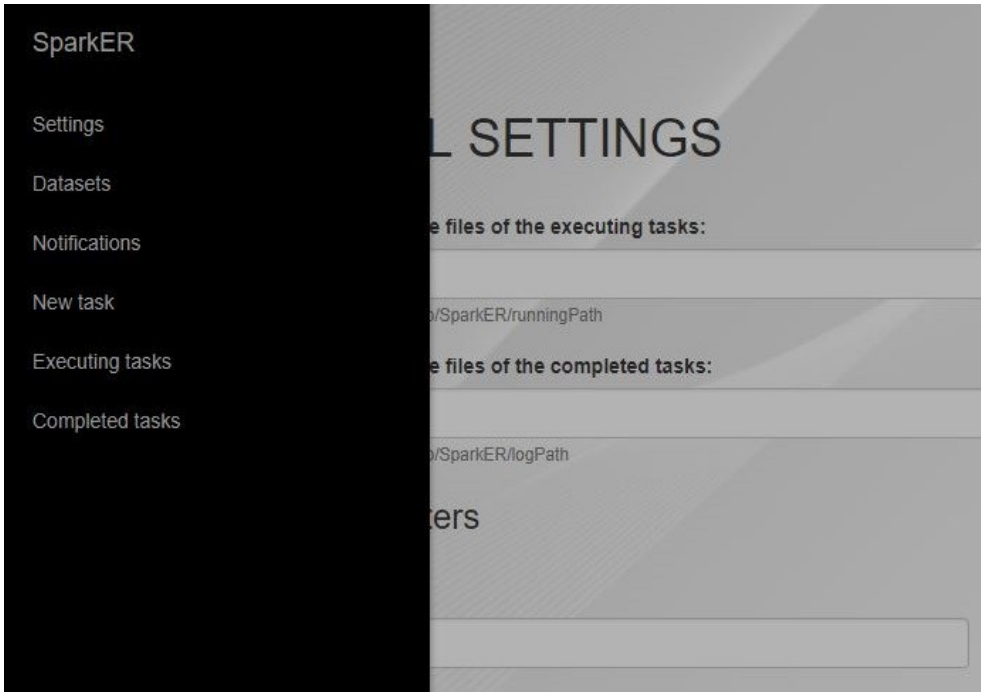
In seguito alla decisione di usare AngularJS, il sito web è stato strutturato come una Single-page application. Vi è quindi una sola pagina statica, chiamata "index.html" con tutti i riferimenti agli script e ai file css necessari e nella quale è presente il contenitore per le viste (definito dalla direttiva *ng-view*). Tutti i file accessori per il sito web sono nella cartella "static". Tra essi troviamo:

- la cartella "partials" in cui vi sono i template HTML che definiscono le viste;
- le cartelle con tutte le librerie css e gli script per l'utilizzo di Bootrap, HighCharts e dei moduli di Bootstrap;
- lo script "angularScript.js", in cui vi è il codice Javascript specifico per l'applicazione che contiene la definizione del modulo dell'applicazione. In questo modulo sono definiti i controller e la configurazione del routing.

Attraverso il servizio *$routeProvider* ad ogni template è stato associato un controller e un URL relativo "virtuale". Per poter navigare verso una vista, è necessario inserire nella pagina principale il suo URL virtuale in un collegamento ipertestuale preceduto dal simbolo "*#!*". Quando quel collegamento viene premuto, verrà immediatamente cercato il template corrispondente e inserito nel contenitore per le viste sulla pagina principale senza caricamento completo di pagina. Sulla barra di navigazione comparirà il seguente URL: "URL del dominio" + "*#!*" + "URL virtuale della vista". Nell'applicazione per SparkER, i collegamenti con le viste sono stati tutti inseriti in un menù di navigazione scorrevole definito nella pagina principale.

```
app.config(function($routeProvider){
    $routeProvider.when("/settings", {
        templateUrl: "/static/partials/settings.html",
        controller: "settingsController"
    });

    $routeProvider.when("/datasets", {
        templateUrl: "/static/partials/datasets.html",
        controller: "datasetsController"
    });

    $routeProvider.when("/notifications", {
        templateUrl: "/static/partials/notifications.html",
        controller: "notificationsController"
    });

    $routeProvider.when("/newTask", {
        templateUrl: "/static/partials/newTask.html",
        controller: "newTaskController"
    });

    $routeProvider.when("/executingTasks", {
        templateUrl: "/static/partials/executingTasks.html",
        controller: "executingTasksController"
    });

    $routeProvider.when("/completedTasks", {
        templateUrl: "/static/partials/completedTasks.html",
        controller: "completedTasksController"
    });
    $routeProvider.otherwise({redirectTo: "/settings"});
});
```

```
<sidebarjs>
    <ul class="sidebar-nav">
        <li class="sidebar-brand">
            <a href="#!/home"> SparkER</a>
        </li>
        <li>
            <a href="#!/settings">Settings</a>
        </li>
        <li>
            <a href="#!/datasets">Datasets</a>
        </li>
        <li>
            <a href="#!/notifications">Notifications</a>
        </li>
        <li>
            <a href="#!/newTask">New task</a>
        </li>
        <li>
            <a href="#!/executingTasks">Executing tasks</a>
        </li>
        <li>
            <a href="#!/completedTasks">Completed tasks</a>
        </li>
    </ul>
</sidebarjs>
```



In alto a sinistra, il codice di AngularJS che associa i template delle viste ai controller e agli URL virtuali. In alto destra, sorgente della pagina "index.html" che definisce il menù di navigazione del sito web. Si possono notare gli URL virtuali usati nei collegamenti ipertestuali. In basso, una schermata con la barra di navigazione del sito

4.1.2 Struttura del server

Sul server troviamo tutte le risorse necessarie per l'applicazione: una cartella con il jar eseguibile SparkER.jar e le sue librerie, una con tutti i file JSON necessari per il salvataggio dei dati e una con tutti i dataset su cui poter effettuare i task di ER. Il codice lato server si trova in uno script Python chiamato "main.py" ed è integrato con il codice del framework Flask per il server web.

Quest'ultimo definisce una sola funzione per servire l'unica pagina web che compone il sito, ("index.html"), mentre tutte le altre funzioni definite sono associate a un URL, attraverso il quale queste funzioni sono accessibili alle chiamate AJAX. Queste sono le funzioni attraverso le quali il client può accedere ai servizi del server, come il lancio di un nuovo task, la raccolta dei risultati di task dal file di log, la raccolta delle corrispondenze trovate con il meta-blocking, ecc. Non è stato utilizzato l'approccio ad oggetti, ma solo quello della programmazione strutturata. Nello script vi è anche il collegamento al bot di telegram che si chiama "SparkER" ed è stato incluso in una chat di gruppo, "SparkERGroup", assieme a tutti gli utenti a cui si vuole rivolgere i risultati dei test.

```python
from flask import Flask, render_template, request, jsonify
import subprocess, json, os, re, telepot, glob,  pandas
from threading import Thread

bot = telepot.Bot("Token of Telegram Bot")

app = Flask(__name__)

@app.route('/')
def index():
    return render_template('index.html')

@app.route('/changeSettings', methods=['POST'])
def changeSettings():

    f = open("static/JSON_files/settings.json", "w")

    data = request.get_json()
    json.dump(data, f)
    f.close()
    return "Settings changed"


@app.route('/changeDatasetsList', methods=['POST'])
def changeDatasetsList():

    f = open("static/JSON_files/datasets.json", "w+")

    data = request.get_json()
    json.dump(data, f)
    f.close()
    return "Dataset added/deleted"

```

Codice sorgente del server web dell'applicazione contente la funzione per servire la pagina e due delle funzioni a cui il decoratore *@app.route* associa un URL, rendendole accessibili ad AJAX. Da notare anche il collegamento al bot di telegram con la funzione *Bot* di telepot, la quale prende in input il token del bot.

4.2 FUNZIONALITÀ DEL SISTEMA

4.2.1 Inserimento o modifica delle impostazioni generali

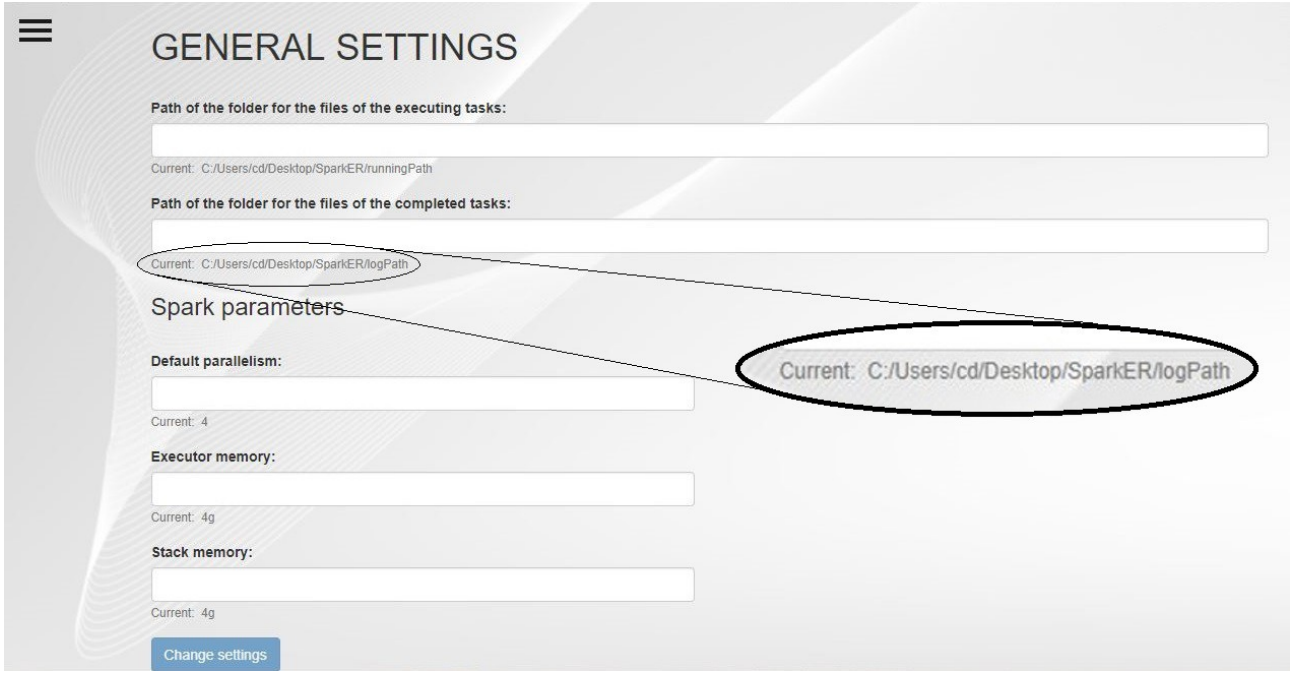


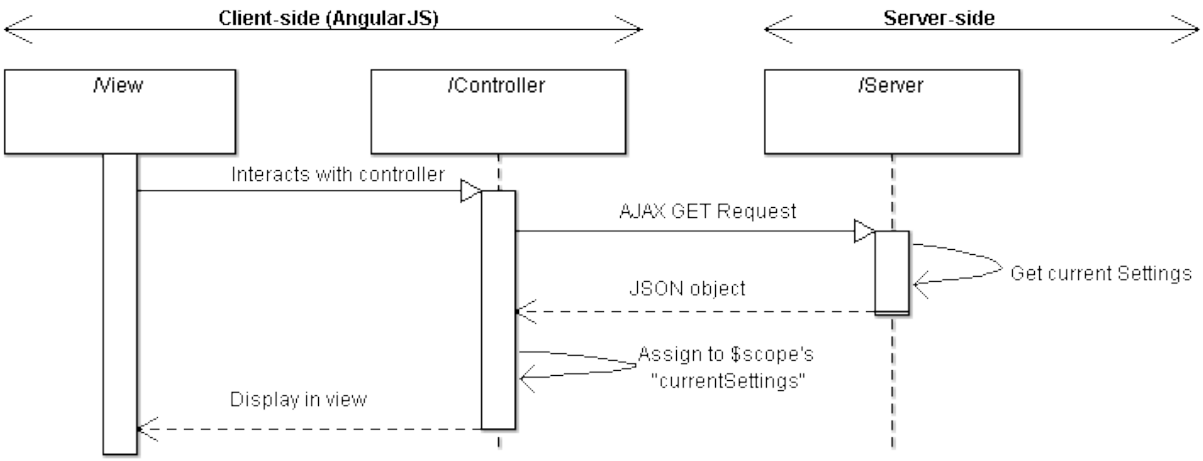
Immagine della schermata per l'inserimento delle impostazioni generali

La prima delle viste riguarda cinque dei parametri che SparkER deve prendere in ingresso:

- "runningPath", il percorso assoluto della cartella in cui viene messo il log di un task durante la sua esecuzione;
- "logPath", il percorso assoluto della cartella in cui viene messo il log di un task alla fine della sua esecuzione, assieme ai file del meta-blocking;
- "default Parallelism", il livello di parallelismo;
- "executorMemory", memoria dell'esecutore;
- "stackMemory", memoria delle stack.

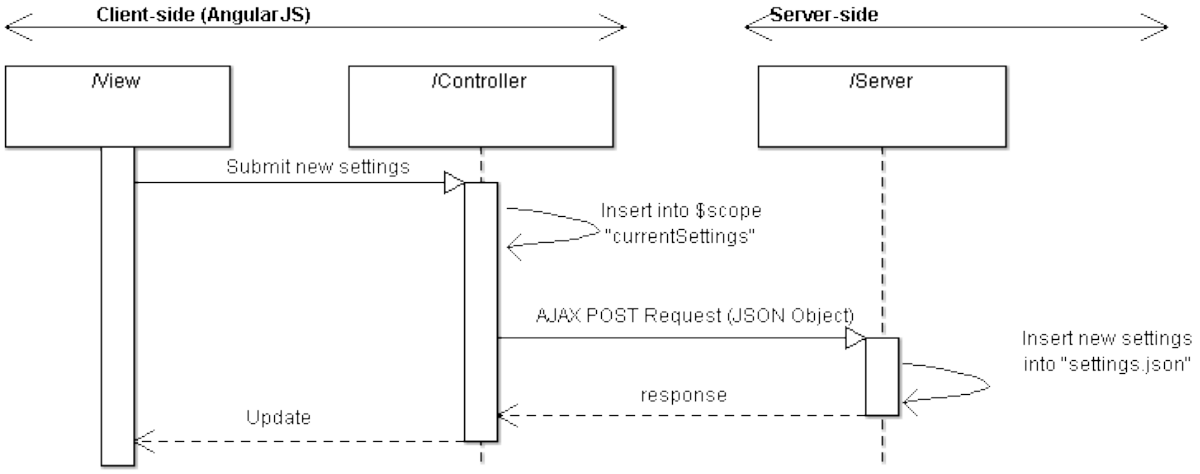
Si tratta di parametri generali che riguardano l'impostazione dell'ambiente di lavoro, per questo vengono chiamati "impostazioni generali".
Gli ultimi tre possono necessitare di modifiche se l'applicazione viene spostata su una macchina con caratteristiche architetturale molto diverse da quella in cui risiede. Il parametro "sparkLocalDir", il quale indica la cartella provvisoria di lavoro di Apache Spark può rimanere uguale qualsiasi sia la macchina di esecuzione.

Al di sotto di ogni campo di input è presente un'etichetta con l'impostazione corrente del paramentro. Ogni volta che si naviga verso questa vista, il controller associato ad essa effettua una richiesta AJAX di tipo GET che prende le impostazioni correnti dei parametri dal server, che si trovano in un file JSON chiamato "settings.json". Inserisce poi queste impostazioni nel modello dei dati, in un oggetto dello scope chiamato *currentSettings*, rendendole visualizzabili nella vista.



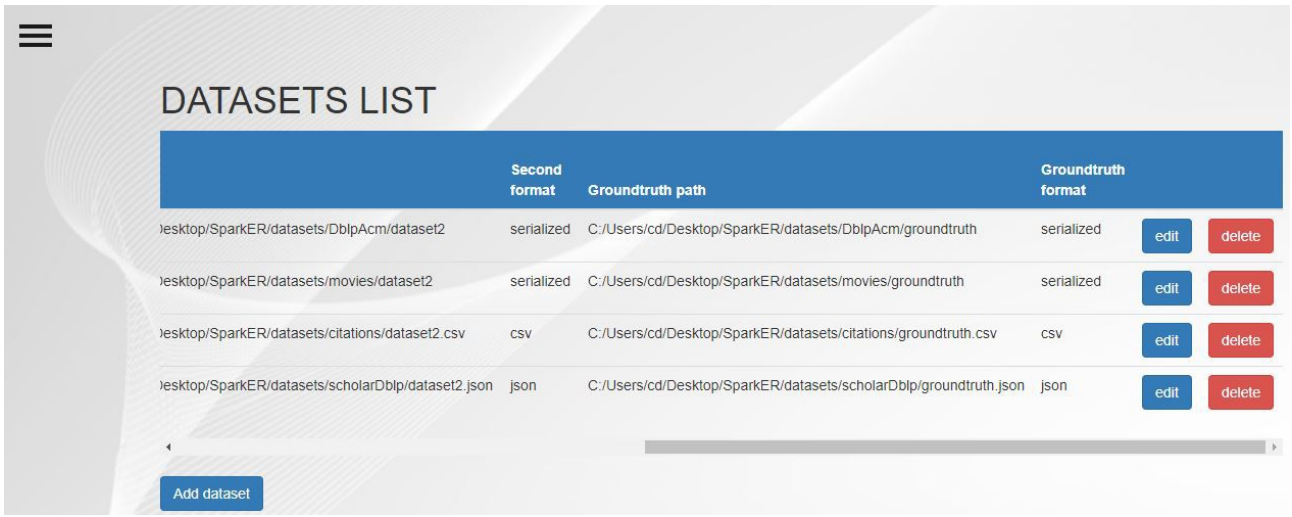
Sequence Diagram sul il reperimento delle impostazioni correnti

La modifica delle impostazioni avviene segue questa procedura: l'utente inserisce le nuove impostazioni nei campi di input del form; nessun campo può essere lasciato vuoto, altrimenti il pulsante di sottomissione rimane disattivato; i valori inseriti sono automaticamente associati alle proprieta di un oggetto JSON chiamato *newSettings*; esso viene passato tramite richiesta AJAX (POST) alla una funzione del server chiamata *changeSettings* ,il quale lo inserisce nel file "settings.json".
Ogni volta che vengono cambiate le impostazioni, viene automaticamente rifatta la richiesta delle impostazioni correnti, in modo da poter aggiornare le etichette sotto i campi di input.



Sequence diagram sulla modifica delle impostazioni

4.2.2 Inserimento, modifica o eliminazione di un dataset



Immangine della schermata che mostra la lista dei dataset inseriti

Nella seconda vista è presente una tabella con nove possibili campi, otto dei quali corrispondono ai parametri dati in input a SparkER che riguardano il dataset su cui eseguire un task di ER; il nono corrisponde al nome dato al dataset. La lista dei parametri dei dataset è mantenuta in un file sul server chiamato "datasets.json", dal quale viene presa con una richiesta AJAX di tipo GET, ogni volta che si naviga verso questa vista. La lista viene inserita nel modello dei dati in una variabile chiamata *datasetsList*, sulla quale la tabella si costruisce attraverso la direttiva *ng-repeat*.

```
<tbody>
    <tr ng-repeat="x in datasetsList">
        <td>{{x.name}}</td>
        <td>{{x.datasetRealProfileID}}</td>
        <td>{{x.datasetType}}</td>
        <td>{{x.dataset1Path}}</td>
        <td>{{x.dataset1Type}}</td>
        <td><div ng-if="isClean(x.datasetType)">{{x.dataset2Path}}</div></td>
        <td><div ng-if="isClean(x.datasetType)">{{x.dataset2Type}}</div></td>
        <td>{{x.groundtruthPath}}</td>
        <td>{{x.groundtruthType}}</td>
        <td><a ng-click="changeDatasetsList('edit', x)" class="btn btn-small btn-primary">edit</a></td>
        <td><a ng-click="changeDatasetsList('delete', x)" class="btn btn-small btn-danger">delete</a></td>
    </tr>
</tbody>
```

Codice sorgente del body della tabella dei dataset

Tramite la sintassi *ng-repeat="x in datasetsList"*, viene inserita automaticamete una riga nella tabella per ogni elemento della lista dei dataset; perciò, per aggiungere una riga sulla vista, basta semplicemente aggiungere un elemento alla variabile *datasetsList*. Questo è quello che viene fatto ogni volta che viene aggiunto un dataset. La procedura segue questi passi:

1) Quando viene premuto il pulsante "Add dataset", compare una finestra modale (o "popover") nella quale è presente un form;
2) Il form serve per passare al controller i parametri del nuovo dataset. Sono necessari il nome, il tipo di dataset (Clean o Dirty), il percorso assoluto e il formato del dataset (in caso tipo Clean, nella stessa riga tali parametri andranno definiti per due dataset) ed il percorso e il formato dell'eventuale groundtruth. Il bottone di sottomissione sarà disattivato finché i parametri richiesti non saranno inseriti e validi;

3) Il parametri inseriti sono automaticamente associati alle proprietà di un oggetto chiamato *newDataset*, il quale verrà aggiunto alla lista dei dataset, comparendo automaticamente sulla vista;
4) Il controller prende l'intera lista dei dataset e la manda al server con una richiesta AJAX di tipo POST. La lista sovrascriverà quella precedente nel file "datasets.json".

Questa operazione non prevede l'uploading del dataset vero e proprio, infatti, i datasets sono spesso molto grandi ed è necessario che siano già sul server.

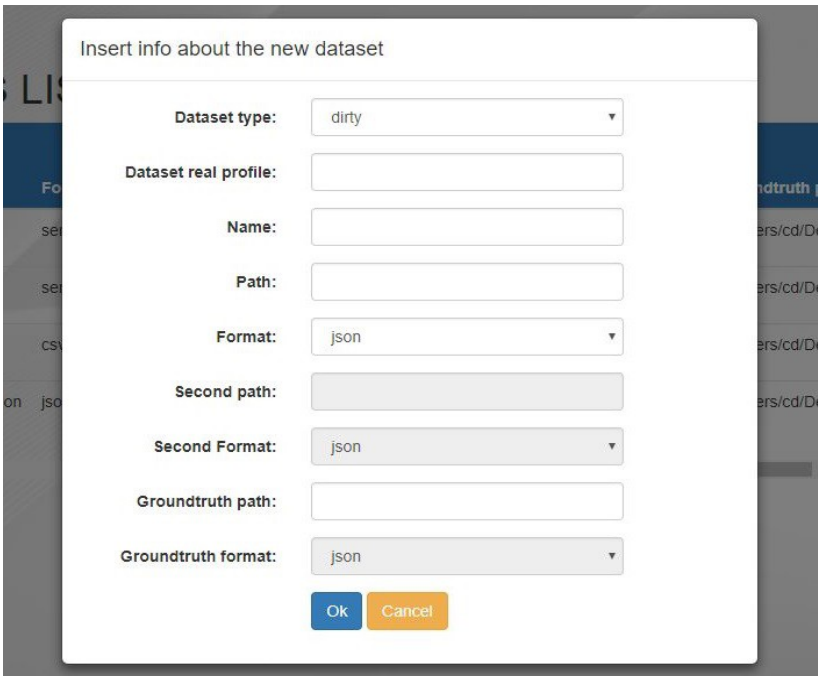
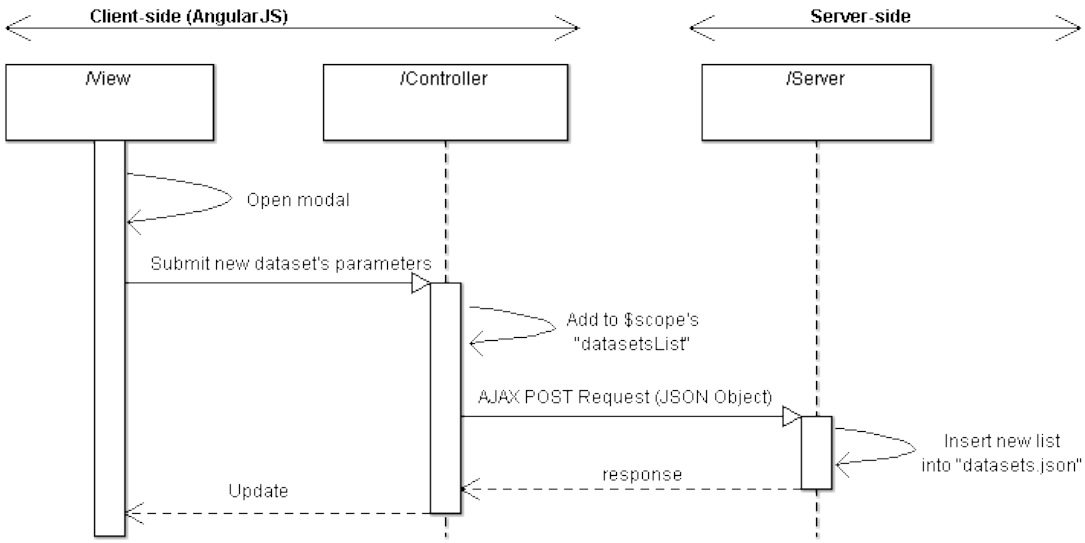


Immagine della finestra modale per l'aggiunta di un dataset



Sequence diagram per l'aggiunta di un dataset

Ogni riga ha due pulsanti alla fine ("edit" e "delete") che servono per la modifica e l'eliminazione

dei parametri del dataset dalla lista. Per la modifica ricomparirà la finestra modale con i campi di input contenenti i valori correnti dei parametri del dataset. L'eliminazione prevede semplicemente che l'elemento corrispondente alla riga venga tolto dal dataset. In entrambi i casi, il file "dataset.json" verrà aggiornato allo stesso modo del processo di aggiunta.

4.2.3 Lancio di un nuovo task

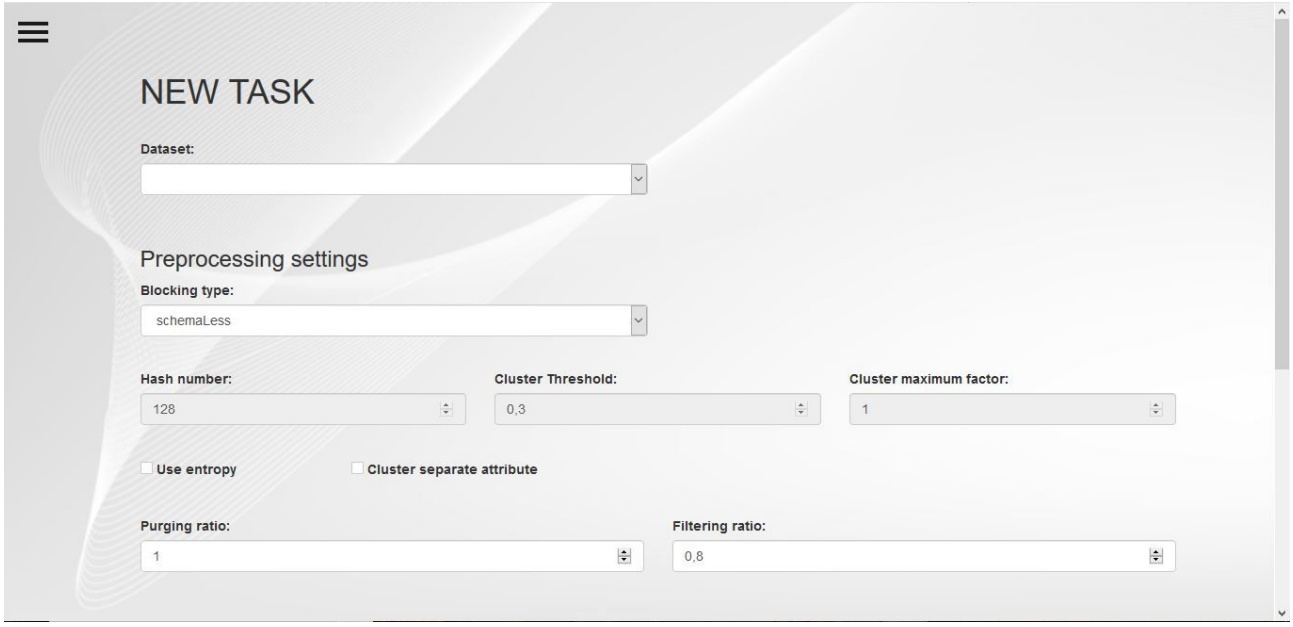


Immagine della schemata per il lancio di un nuovo task

La terza vista è quella per il lancio di un nuovo task. Essa contiene un form per l'inserimento dei parametri riferiti specificatamente al task.
Quando si naviga verso questa vista, il controller effettua una chiamata AJAX per recuperare la lista dei parametri dei dataset dal file "datasets.json"; questa lista viene inserita in una proprietà dello scope chiamata *datasetList*. I nomi dei dataset vengono poi usati come valori delle opzioni della prima select in alto. Tramite questa select è possibile scegliere il dataset (o coppia di dataset in caso di tipo Clean) su cui effettuare l'ER.
Vi sono poi i seguenti campi di input:

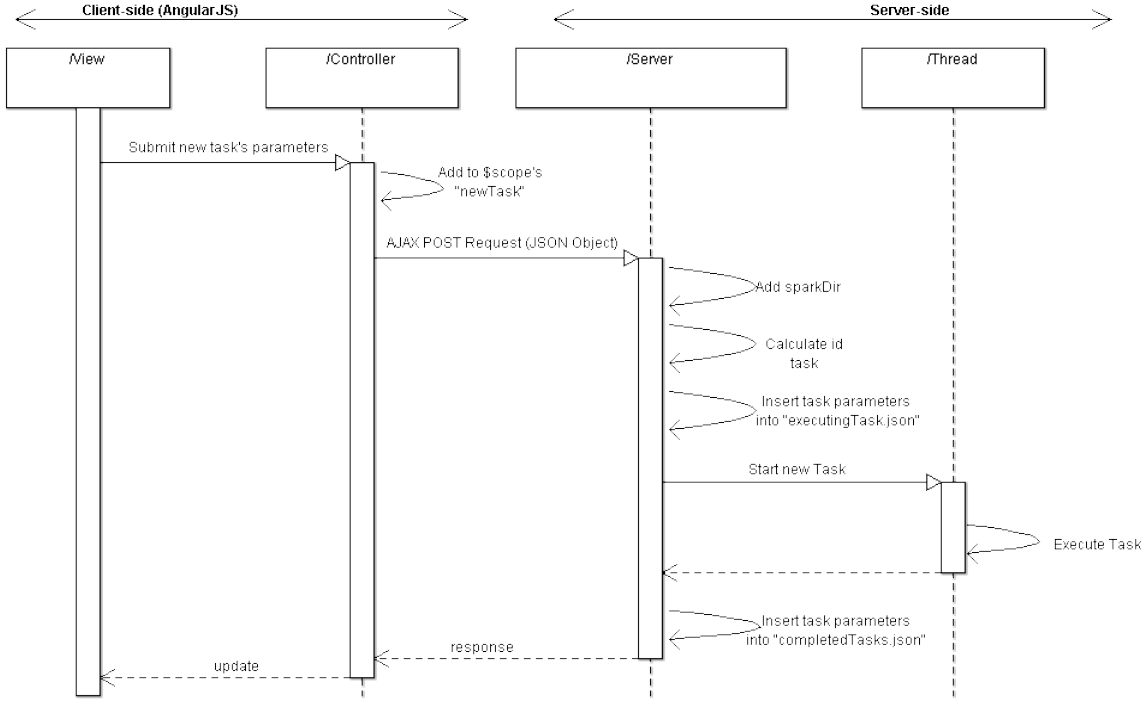
- una select per il "blockingType", il parametro che indica il tipo di blocking (Token Blocking o Loose Schema Blocking);
- cinque campi per i parametri per l'impostazione del clustering (in caso di Loose Schema Blocking);
- un campo di testo per la "purgingRatio", paramentro per la fase di purging (può essere solo un numero);
- un campo di testo per la "filteringRatio", paramentro per la fase di filtering (può essere solo un numero tra 0 e 1);
- una select "pruning schema", schema di pruning (avg o maxdiv2);
- un campo di testo per il "clusterThreshold", soglia di clustering in caso sia stato scelto "avg" come schema di pruning;

- una select per il "pruningType", tipo di pruning (and o or);
- una select multipla"weightTypes", metodi di pesatura degli archi nel grafico di meta-blocking. Se ne può scegliere uno o più;
- un campo di testo per il "chi2Divider", parametro necessario al metodo di pesatura "chiSquare" in caso venga scelto (può essere solo un numero intero);

Il form fa un uso massiccio delle direttive AngularJS in quanto molti parametri dipendono dal valore di altri. Ad esempio, in caso venga scelto il Token blocking, i cinque campi di input per i parametri del clustering vengono disabilitati, altrimenti sono abilitati e obbligatori. Il pulsante per l'invio dei dati rimane disattivato fino a quando i parametri richiesti non sono stati inseriti e sono corretti.

Il lancio di un nuovo dataset ha la seguente procedura:

1) Una volta premuto il pulsante, i parametri del dataset scelto, del preprocessing e del meta-blocking vengono inseriti in un oggetto JSON chiamato *newTask*. Vengono poi mandati via AJAX a una funzione del server;
2) La funzione aggiunge nell'oggetto *newTask* i parametri delle impostazioni generali (presi dal file "settings.json"), il parametro "sparkLocalDir" (definito nel codice) e l'id del task. Quest'ultimo viene calcolato come il massimo id dei task già esistenti (terminati o in esecuzione) aumentato di uno;
3) Viene eseguito un thread separato dal processo principale in cui viene lanciato il nuovo task, attivando il jar eseguibile del framework SparkER e passandogli in input l'oggetto JSON con tutti i parametri;
4) L'oggetto JSON viene inserito anche nel file "executingTasks.json", il quale contiene la lista dei parametri di tutti i task in esecuzione. Quando un task termina il task, la lista dei suoi parametri viene spostata nel file "completedTasks.json".



Sequence diagram per il lancio di un nuovo task

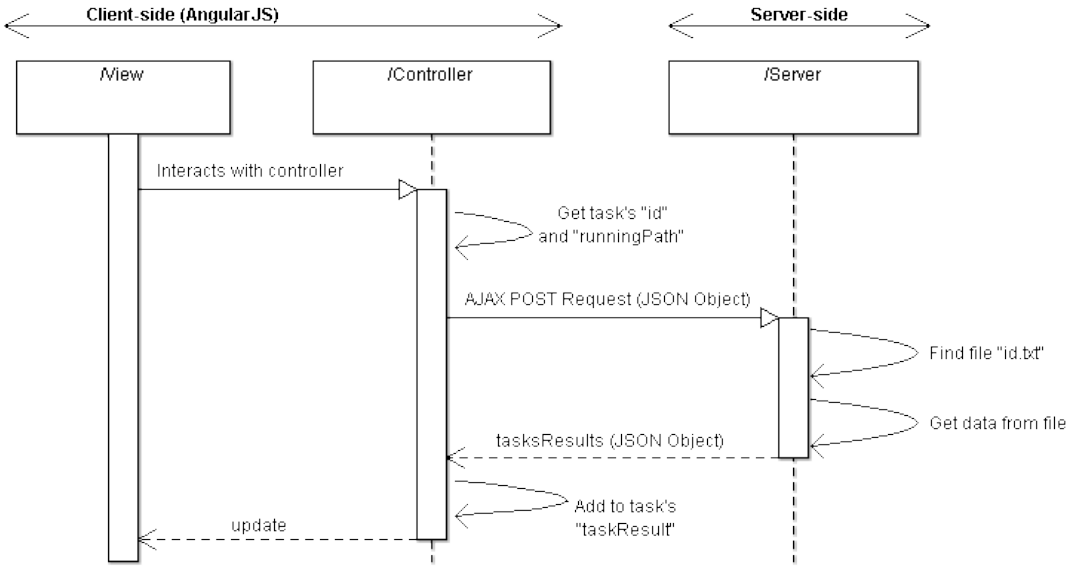
4.2.4 Monitoraggio di un task in esecuzione

Schermata dei task in esecuzione

La quarta vista è quella dedicata al monitoraggio dei task in esecuzione. È composta da una tabella con tre campi, relativa all'id del task, al nome del dataset usato da quel task, allo stato di esecuzione del task. Appena si naviga verso questa vista, il controller associato ad essa effettua una chiamata AJAX per prendere la lista dei parametri dei task in esecuzione dal file "executingTasks.json". Questa lista viene immessa in una variabile dello scope chiamata *tasksList* e la tabella viene costruita su questa lista tramite la direttiva *ng-repeat.*
Se la lista contiene degli elementi, allora vuol dire che ci sono dei task in esecuzione. Questi verranno monitorati attraverso un ciclo sulla lista che viene effettuato automaticamente ogni due secondi. All'interno del ciclo vi sono i seguenti step:

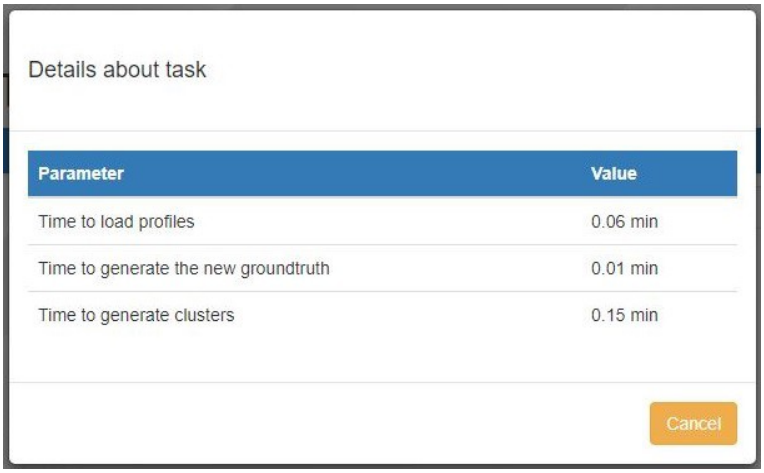
1) Viene effettuata una chiamata AJAX di tipo POST a un metodo del server chiamato *infoTask*, il quale riceve dal client l'identificatore del task ("id") e il percorso assoluto della cartella dei log durante l'esecuzione.
2) Tramite i due parametri può risalire al file di log del task corrispondente poiché esso si chiama "id.txt". Una volta trovato, su di esso vengono effettuate delle operazioni dette "regex". Queste ultime permettono di trovare nel file le stringhe contenenti i risultati richiesti del task e di assegnarli a delle variabili prestabilite. Queste variabili divengono tutte proprietà di un oggetto JSON che verrà mandato al client.
3) Il controller riceve il JSON con i risultati del task e li inserisce nella lista dei suoi parametri in una proprietà chiamata *taskResults*. Questo li renderà visualizzabili sulla vista.

Per poter capire in che fase di esecuzione si trova il task, si guarda qual'è l'ultima proprietà con un valore tra le proprietà dell'oggetto *taskResults*. Durante una determinata fase, sul log saranno presenti soltanto le stringhe con i risultati relativi alle fasi precedenti. Perciò, quando vengono effettaute le regex, alle proprietà associate alla fase corrente e a quelle successive non sarà dato alcun valore poichè quelle fasi devono ancora avvenire. La fase in cui si trova il task è la fase successiva a quella a cui si riferisce l'ultimo attibuto con valore. Ad esempio, se scorrendo le proprietà troviamo che l'ultima proprietà con valore è "blockingTime" (tempo di blocking), significa che la fase di blocking è terminata e che il task è in fase di purging.

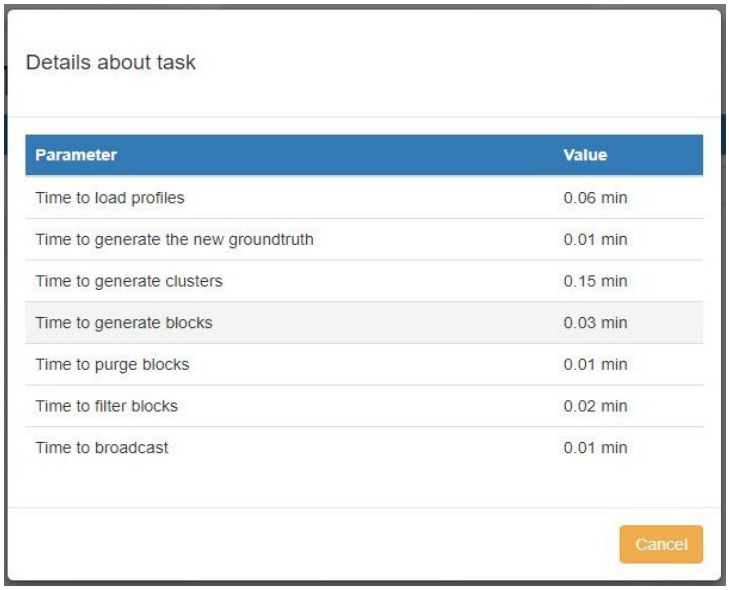


Sequence diagram per il raccoglimento dei risultati di un task

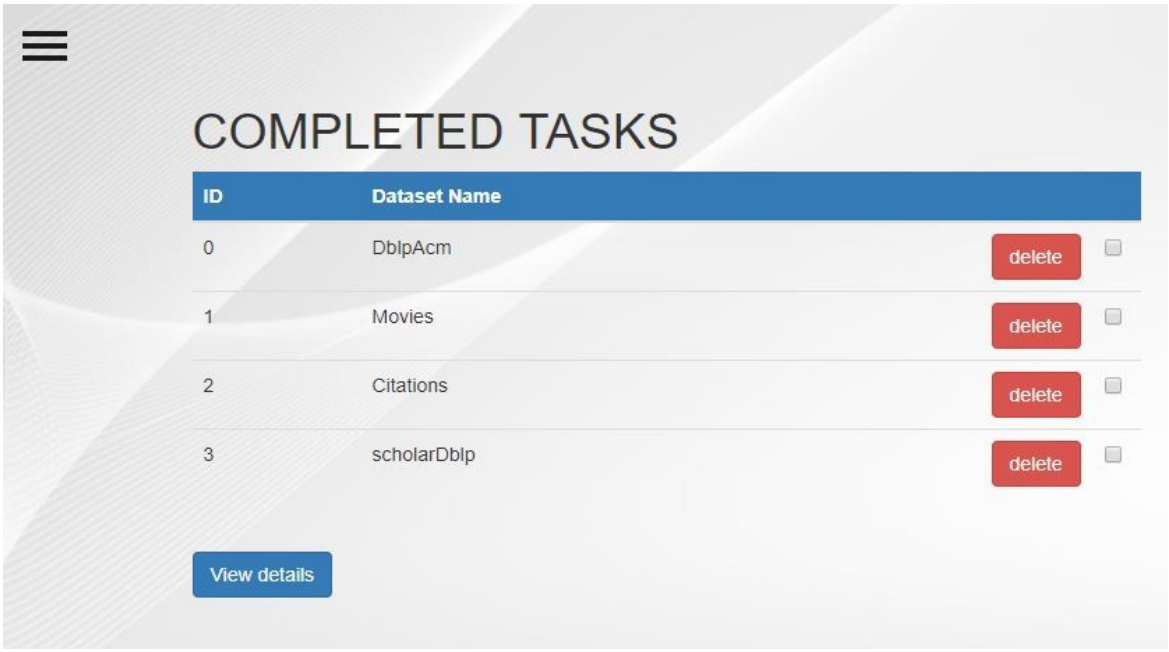
Per ogni task in esecuzione è possibile visualizzare nel dettagli i tempi di esecuzione delle singole fasi attraverso il pulsante "View details", il quale apre una finestra modale che mostra questi tempi approssimati al centesimo. Chiaramente in base alla fase in cui si trova il task, saranno presenti più o meno dettagli.



In alto, finestra modale con i tempi di un task in fase di blocking. A destra, i tempi dello stesso task ma in fase di meta-blocking; ci sono più risultati poiché si trova in una fase successiva



4.2.5 Visione e confronto dei risultati dei task terminati

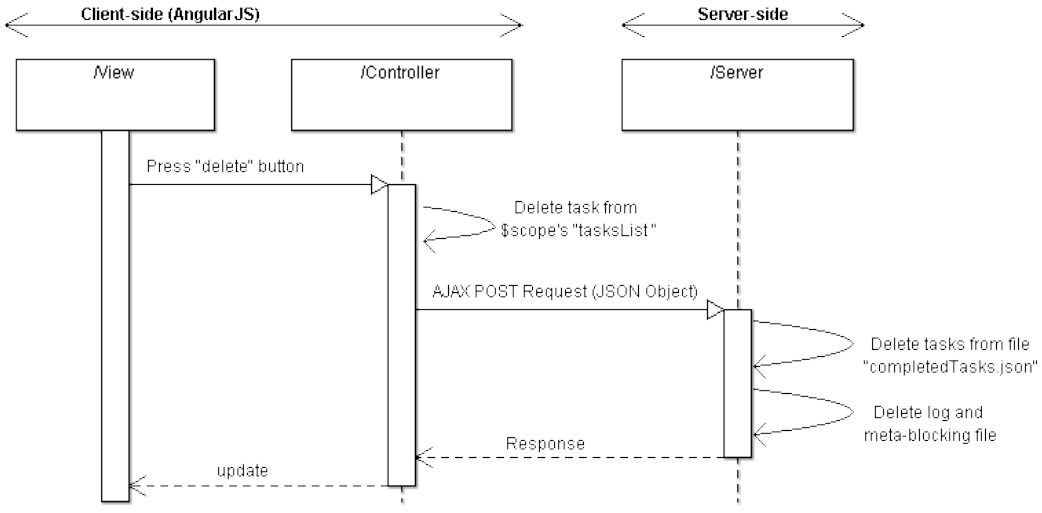


Schermata con la lista dei task terminati

La quinta vista è quella per la visione dei risultati dei task. Anch'essa è composta da una tabella la quale si costruisce su una variabile dello scope chiamata *taskList* tramite la direttiva *ng-repeat.* La lista viene aggiornata automaticamente quando si naviga verso questa vista, sempre tramite una richiesta AJAX di tipo GET, la quale stavolta prende i dati dal file "completedTasks.json". Successivamente, per ogni task terminato, vengono presi i risultati nella stessa maniera della vista sui task in esecuzione. Stavolta, però, l'operazione viene fatta una volta poiché i task sono, appunto, terminati.

La tabella contiene l'id del task e il nome del dataset su cui è stato effettuato il task, un pulsante per l'eliminazione dei dati del task e un checkbox. La procedura di eliminazione dei dati di un task prevede che vengano eliminati:

- i parametri di configurazione del task presenti nella lista del file "completedTasks.json";
- il file di log e i file con i risultati del metablocking;



Sequence diagram per l'eliminazione dei dati di un task

Ogni riga della tabella ha un checkbox il quale, se selezionato, permette di visualizzare i risultati relativi al task associato alla riga. I risultati verranno mostrati nella finestra modale che comparirà premendo il pulsante "View details".

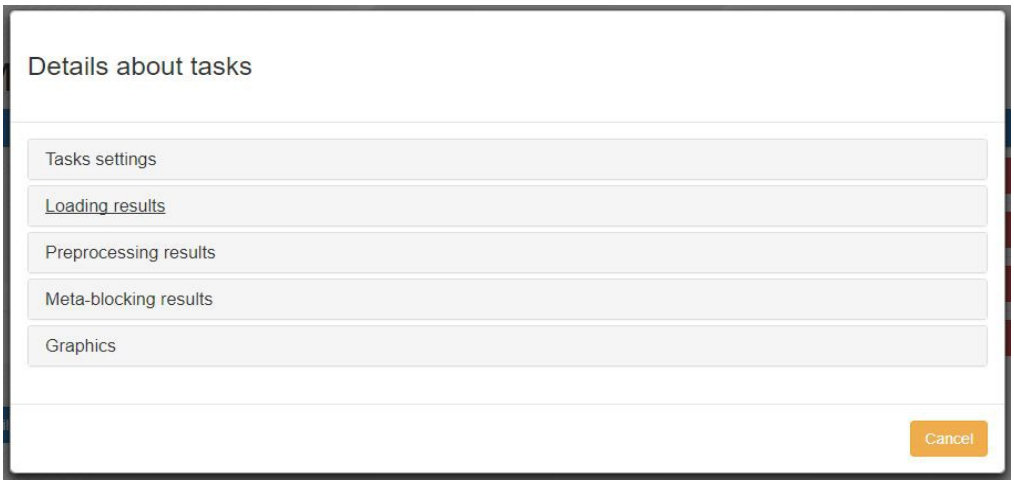
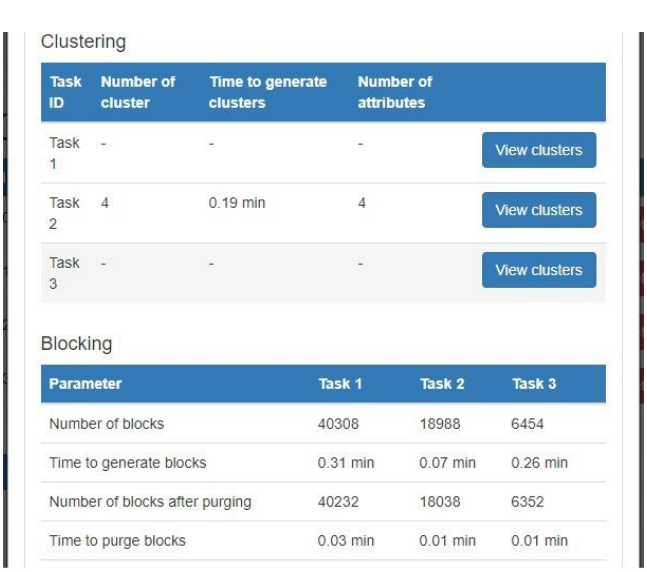


Immagine dela finestra modale

La finestra contiene cinque 'barre a fisarmonica', se premute mostrano una sezione di dettagli relativi a un task. Le sezioni sono:

- **Task settings**, che contiene i parametri di configurazione di un task al momento del lancio. Sono inseriti in una tabella con una colonna per ogni task;
- **Loading results** che riguardano i risultati della fase durante la quale sono stati individuati i profili del dataset. Tra questi, vi sono il tempo di durata della fase e il numero i profili trovati nel dataset. Sono inseriti in una tabella con una colonna per ogni task;
- **Preprocessing results**, tra i quali vi sono i tempi delle fasi di blocking, purging e filtering e i blocchi esistenti dopo queste fasi. È presente anche una tabella con tante righe quanti sono i task selezionati. Ogni riga contiene i dati sull'eventuale clustering eseguito nel task (se non è stato eseguito, la riga sarà vuota) e vi è anche un bottone "View clusters" che apre una nuova finestra che contiene i dettagli sui clusters generati da quel task (se non è stato effettuato il clustering, la finestra sarà vuota);



A sinistra, immagine delle tabelle riguardanti il preprocessing. Sopra, la finestra modale con i dettagli sul clustering

- **Meta-blocking results**. Per ogni task selezionato vi è una tabella con tante righe quanti sono i metodi di pesatura degli archi scelti in quel task, per ognuno dei quali è stato fatto il meta-blocking. Ogni riga contiene i dati generali relativi al meta-blocking effettuato con il corrispondente metodo di pesatura (numero di archi mantenuti, corrispondenze trovate, PC, PQ e tempo di esecuzione). È presente in ogni riga anche un bottone "View Records" che apre una nuova finestra modale in cui vengono mostrate le coppie di record corrispondenti trovate con quello specifico meta-blocking. In questa finestra ne verranno caricate venti alla volta su richiesta dell'utente. La cosa è possibile solo per i task che hanno utilizzato dataset in formato json o csv.

Task 2

Weight type	Number of retained edges	Number of matches found	PC	PQ	Metablocking time	
js	5024.0	4975	1	0.99	0.01 min	View records
chiSquare	5069.0	4984	1	0.98	0.05 min	View records
arcs	3861.0	3835	0.77	0.99	0.02 min	View records

Sopra, la tabella con i dati sul meta-blocking relativi a un task

- **Graphics**, sezione contente tre grafici (realizzati con highcharts), uno per il tempo di esecuzione, uno per PC e uno per PQ. Sono grafici a colonne in cui l'asse x è diviso in sei parti, una per ogni metodo di pesatura. Per ogni metodo compariranno tante colonne quanti sono i task che hanno usato quel metodo per il meta-blocking, ognuna delle quali indicherà il valore del parametro esaminato per quello specifico task con quel specifico metodo di pesatura.

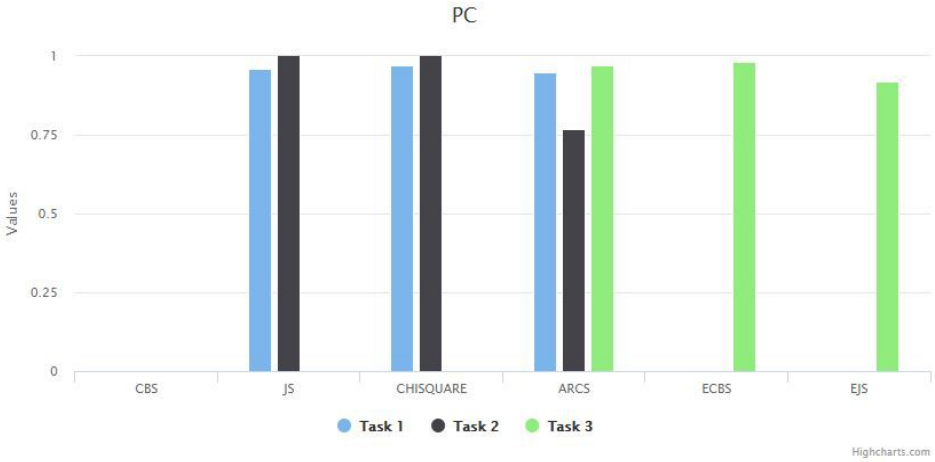
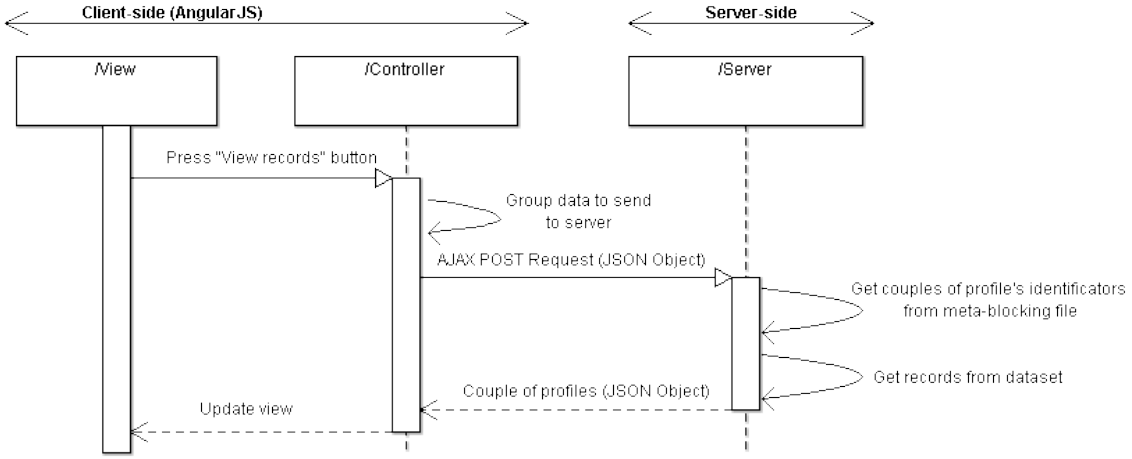


Grafico che mostra i valori del PC per tre task, ognuno dei quali ha usato diversi metodi di pesatura. Le colonne dello stesso colore sono riferite allo stesso task

La procedura di recupero dei records è la seguente:

1) Una volta premuto il bottone "View Records", viene attivata una richiesta AJAX di tipo POST in cui vengono inviati al server i parametri del task necessari per trovare il file di meta-blocking corrispondente (identificatore del task, metodo di pesatura e percorso assoluto della cartella dove si trova il log) assieme al percorso assoluto e al formato del dataset (dei due dataset in caso di tipo Clean);
2) La funzione che riceve la richiesta si chiama *getRecords*, la quale, una volta trovato il file di meta-blocking corrispondente, recupera le prime venti coppie di identificatori dei profili corrispondenti. Successivamente, tramite gli identificatori, risale ai profili veri e propri all'interno del dataset e li mette in una lista di coppie. Ogni elemento di una coppia è costituito all'interno dal record e dal suo identificatore;
3) La lista di coppie viene restituita come oggetto JSON al controller della vista, il quale le rende visibili in serie sulla nuova finestra modale aperta.

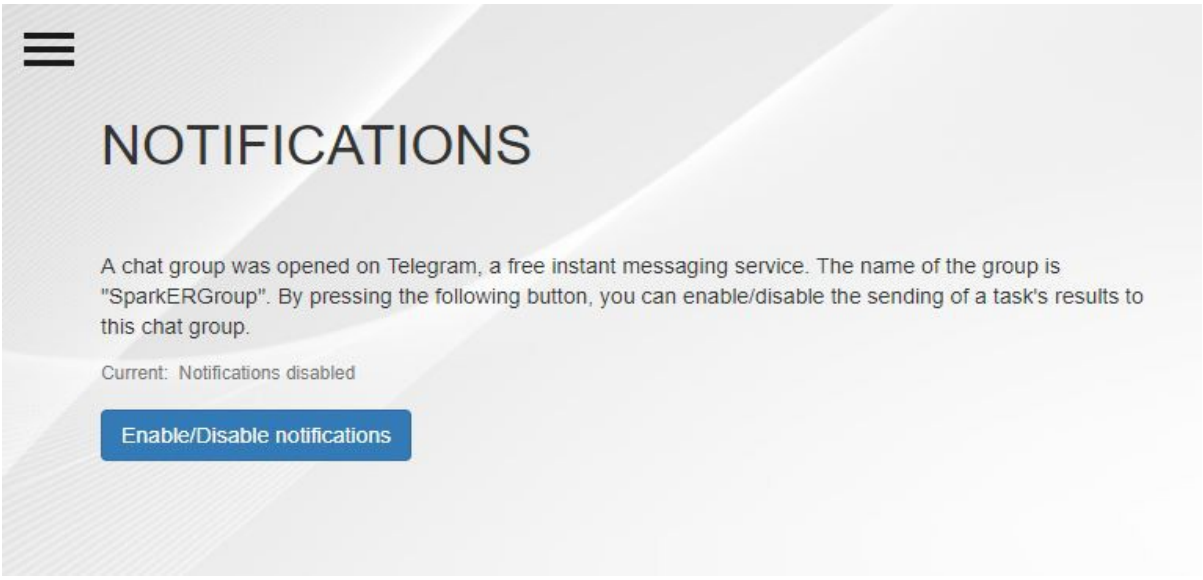



Records

1301413	title : 'On Blocking Zeros and Strong Stabilizability of Linear Multivariable Systems*t' , authors : 'Ben M. Chen, P. Sannuti'
2323376	title : 'On blocking zeros and strong stabilizability of linear multivariable systems.' , authors : 'Ben M. Chen, Ali Saberi, Peddapullaiah Sannuti' , journal : 'Automatica' , year : '1992' , publication_type : 'article'
878310	title : 'Simple-Cell-Like Receptive Fields Maximize Temporal Coherence in Natural Video' , authors : 'Jarmo Hurri, Aapo Hyvrinen' , year : '2002.0'
1735532	title : 'Simple-Cell-Like Receptive Fields Maximize Temporal Coherence in Natural Video.' , authors : 'Jarmo Hurri, Aapo Hyvrinen' , journal : 'Neural Computation' , year : '2003' , publication_type : 'article'
1620592	title : 'CUDA-lite: Reducing GPU Programming Complexity' , authors : 'Sain-zee Ueng, Melvin Lathara, Sara S. Baghsorkhi, Wen-mei W. Hwu'
533276	title : 'CUDA-Lite: Reducing GPU Programming Complexity.' , authors : 'Sain-Zee Ueng, Melvin Lathara, Sara S. Baghsorkhi, Wen-mei W. Hwu' , year : '2008' , publication_type : 'inproceedings'

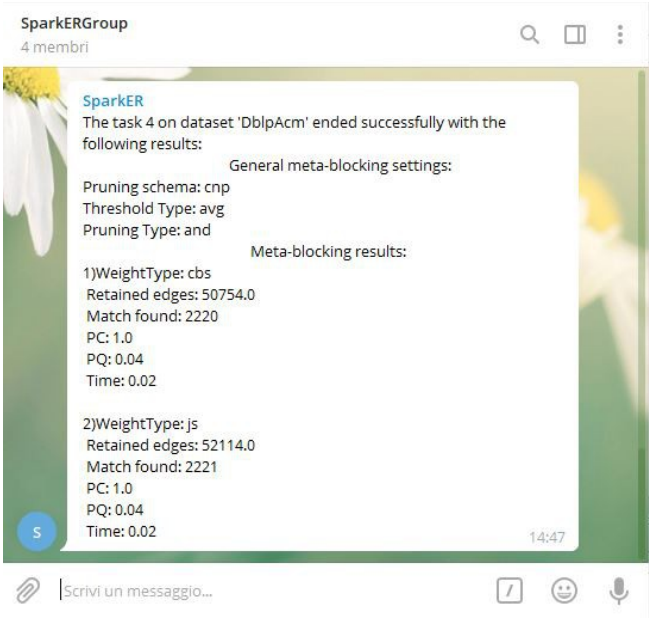
Sopra, sequence diagram per il reperimento di profili corrispondenti. A sinistra, immagine della finestra modale che mostra una serie di profili corrispondenti

4.2.6 Invio di notifiche via Telegram



Schermata per l'impostazione delle notifiche

La sesta vista riguarda l'impostazione delle notifiche da mandare tramite Telegram all'utente. Inizialmente si era pensato alla possibilità di inserire da interfaccia i numeri di telefono degli utenti a cui mandare le notifiche nelle loro chat private. Questa opzione non è stata realizzabile per il seguente motivo: il programma, per poter inviare una notifica tramite il bot, deve conoscere il "chat_id" dell'utente, un identificatore univoco dell'account. Per scoprirlo deve essere prima l'utente a mandare una notifica al bot; esso la inoltrerà al programma sotto forma di oggetto con alcune proprietà, tra le quali vi è il suo chat_id. La specifica originale non prevede che il bot riceva notifiche da utenti. Si è quindi deciso di creare un gruppo su Telegram chiamato "SparkERGroup" nel quale è stato inserito il bot assieme alle persone a cui si vuole rivolgere i risultati del task. Dal gruppo è stata inviata una notifica che è stata ricevuta da uno script Python che l'ha visualizzata in output il "chat_id" del gruppo. Una volta trovato l'identificatore del gruppo, questo è stato inserito all'interno del codice del programma. Ogni volta che un task finisce, viene mandata una notifica solo al gruppo. A seguito di questa variazione, è stata richiesta la possibilità di disattivare o attivare le notifiche. Tramite il pulsante della vista, è possibile fare ciò. Sopra il bottone c'è un'etichetta che mostra l'impostazione corrente.



Notifica di Telegram inviata dal bot alla fine di un task.
In essa sono scritti i risultati del meta-blocking

5 CONCLUSIONI E SVILUPPI FUTURI

Lo sviluppo di quest'applicazione è stata la mia prima esperienza di programmazione web e si è rivelata fruttuosa, in quanto ha permesso di approfondire le conoscenze acquisite nei tre anni del percorso formativo e di acquisirne di nuove.
Quasi tutti i linguaggi usati sono stati studiati in autonomia attraverso la documentazione e le guide presenti su internet e sono stati utilizzati per la prima volta.
L'aspetto più interessante è stato lo studio e l'utilizzo del framework AngularJS, di Boostrap, i quali supportano funzionalità molto interessanti che hanno notevolmente semplificato lo sviluppo di un interfaccia grafica complessa e adattabile a tutti i dispositivi, cosa per cui, normalmente, servirebbero molte righe di codice JavaScript. Tra le funzionalità di AngularJS più apprezzate ci sono quelle del routing di AngularJS e dell'uso di AJAX, che hanno portato a un'applicazione web che offre un'esperienza simile a quella di un'applicazione desktop.
È stato particolamente apprezzato anche lo studio e l'uso di Python, linguaggio molto semplice, flessibile e efficace che si è rivelato un ottimo strumento per lo sviluppo della parte server, grazie anche all'esistenza di un framework semplicissimo come Flask.
Il progetto è stata un'esperienza nuova, difficile e stimolante e ha portato alla risoluzione di un problema complesso e all'acquisizione di competenze e abilità che potranno essere facilmente spendibili in contesti reali.
Tra i principali sviluppi futuri di questo progetto si intravede soprattutto la modifica dell'uso di Telegram, cercando una soluzione più vicina alla specifica orginale. Sono infatti emerse alcune alternative che, se richiesto, potranno essere analizzate. Se il framework SparkER dovesse subire alcune modifiche, potrebbe essere richiesto di modificare l'applicazione di conseguenza.

6 BIBLIOGRAFIA E SITOGRAFIA

[1] Simonini, G., Bergamaschi, S., & Jagadish, H. V. (2016). BLAST: a Loosely Schema-aware Meta-blocking Approach for Entity Resolution. Pvldb, 9(12), 1173–1184.

[2] Papadakis, G., Koutrika, G., Palpanas, T., & Nejdl, W. (2014). Meta-blocking: Taking entity resolution to the next level. IEEE

[3] https://it.wikipedia.org/wiki/HTML

[4] http://www.html.it/guide/guida-html/

[5] https://it.wikipedia.org/wiki/JavaScript

[6] https://it.wikipedia.org/wiki/CSS

[7] https://it.wikipedia.org/wiki/Bootstrap_(informatica)

[8] http://www.html.it/guide/guida-bootstrap/

[9] https://en.wikipedia.org/wiki/Highcharts

[10] https://www.highcharts.com/products/highcharts/

[11] http://www.html.it/guide/guida-angularjs/

[12] https://en.wikipedia.org/wiki/AngularJS

[13] https://it.wikipedia.org/wiki/Form

[14] https://it.wikipedia.org/wiki/Python

[15] https://www.fullstackpython.com/flask.html

[16] http://flask.pocoo.org/

[17] https://it.wikipedia.org/wiki/Telegram

[18] https://core.telegram.org/bots