SocialHub

Introduzione

Il progetto SocialHub è un servizio dedicato alle persone che vogliono cercare informazioni presenti nei socialnetwork utilizzando un singolo portale.

Funzionamento

L'utente che accede al portale può inserire una o più parole chiave e specificare in quale socialnetwork, tra quelli disponibili, effettuare la ricerca ottenendo una lista uniforme di risultati in modo da offrire un panorama generale e raggiungere poi la fonte dei risultati interessati.

Motivazioni del progetto

Attualmente in rete sono presenti moltissimi socialnetwork ciascun con contenuti differenti e anche le varie interfacce per ottenere informazioni da essi sono differenti e variabili nel tempo in quanto cercano di mantenere il loro servizio sempre aggiornato e migliorarlo sia dal punto di vista prestazionale che sul lato sicurezza. Questo però porta a incompatibilità oltre che tra diversi socials anche nelle varie versioni di interfaccia per cui rende molto difficile riuscire a creare un sistema semplice per effettuare un'interrogazione nei vari socials utilizzando un unico strumento che duri nel tempo.

Presentazione del progetto

Per ovviare ai problemi sopra descritti SocialHub concentra tutta l'interfaccia di comunicazione e interrogazione verso i socials all'interno di un servizio offerto da un server implementato in Java che offre poi un'unica interfaccia comune attraverso un protocollo simil-REST, l'implementazione di ciascun servizio consisterà nella creazione di sistemi di connessione indipendenti per ciascun social che possono avvenire anche in tempi differente.

Perché non usare un protocollo REST-FULL

Il motivo per cui si è scelto di non usare un protocollo completamente REST è dato dal fatto che questo non prevede il mantenimento di uno stato per la connessione, mentre SocialHub richiede una connessione per identificare un utente che vuole connettersi ed offrire una cronologia sulle ricerche da lui effettuate in modo che possa accedervi anche in futuro. Se non si mantenesse lo stato della connessione lato server tutta la complessità della gestione di identificazione si sarebbe suddiviso anche lato client che con la scelta effettuata il client deve solo implementare la procedura di autenticazione senza dover conoscere il background effettivo gestito dal server.

Login via OAuth attraverso Twitter

SocialHub permette a un'utente di effettuare il login attraverso Twitter implementato interamente lato server sia per applicazioni web che mobile application mantenuto poi all'interno della sessione attuale

ma non persistente per cui ad ogni nuova sessione l'utente che vuole accedere ai dati precedentemente salvati deve obbligatoriamente effettuare nuovamente la procedura per il login.
Questa scelta semplifica il login via web ma complica la connessione per le applicazioni mobile in quanto richiede maggiori passaggi. è stato scelto questo tipo di login per evitare di gestire sistemi di login che sono già affrontati da Twitter (ed altri) e sfruttare il servizio da loro offerto delegando completamente questo compito.
Inoltre l'uso degli OAuth ha permesso di implementare un sistema di login semplice e sicuro senza la necessità di mantiene informazioni sull'utente che vengono recuperati direttamente da Twitter dopo l'autenticazione. Il server memorizza in una tabella l'ID Twitter solo per motivi tecnici e non verranno mai acceduti direttamente dai clients, e anche se a causa di bugs la tabella diventasse pubblica questa conterrebbe esclusivamente l'ID@twitter che sono di pubblico dominio.

Ricerca

Il sistema di ricerca permette all'utente di inserire delle parole chiave e specificare su quali socialnetworks, tra quelli attualmente disponibili, effettuare la ricerca.
Il servizio restituisce poi i risultati ottenuti in un formato comune comprensivo di un link per raggiungere la sorgente dell'informazione.
Nel caso i parametri di ricerca sono uguali alla richiesta precedente il sistema richiedere nuove informazioni ai socialnetworks in modo trasparente senza effettuare una richiesta esplicita.

Cronologia

Se un utente decide di effettuare il login attraverso Twitter ottiene l'accesso alla cronologia delle ricerche effettuate anche in sessioni precedenti che vengono memorizzati dal servizio e anche sessioni con lo stesso utente condividono in modo sincrono la stessa cronologia.

Progettazione semplificata

Questa parte presenterà una versione semplificato del documento di progettazione che comunque lo descriverà in modo preciso ed accurato.

Casi d'uso

Il servizio deve assicurare i seguenti casi d'uso all'utente

CU0: Login. Permette all'utente di effettuare il login attraverso Twitter.

CU1: Ricerca. Permette all'utente di effettuare una ricerca per parola chiave all'interno dei servizi di interrogazione attualmente disponibili.

CU2: Cronologia. Permette all'utente di visualizzare le interrogazioni precedenti.

Requisiti base

Rispetto al documento precedente è stato tolto il requisito R4 per semplificare la gestione della cronologia che permette solamente la memorizzazione e la lettura in modo seriale.

R1: Persistenza dell'utente.
- Il sistema deve ricordare un utente per poter recuperare i dati a lui associati in sessioni differenti
- Il sistema non deve memorizzare le informazioni sull'utente ad eccezione del'ID assegnato da Twitter per garantire il punto precedente.
- Il sistema deve fornire su richiesta le informazioni account Twitter sull'utente connesso

R2: Persistenza della cronologia per utenti autenticati.
- Il sistema deve garantire la persistenza di parte della sua cronologia nel tempo
- Il sistema deve garantire la persistenza di almeno 50 stringhe di ricerca

R3: Accesso alla cronologia.
- Il sistema deve garantire l'accesso alla sua cronologia memorizzata
- Il sistema deve garantire l'accesso contemporaneo da diverse sessioni di uno stesso utente

R5: Sessioni lunghe non persistenti.
- Il sistema deve mantenere una sessione di almeno 1 ora

R6: Selezione socialnetwork.
- Il sistema deve permettere all'utente di poter selezionare quale dei servizi disponibili utilizzare

R7: Ricerca per parola chiave.
- Il sistema deve permettere la ricerca per una o più parole chiavi.
- Il sistema deve garantire la possibilità di richiedere ulteriori risultati se sono presenti

R8: Accesso alla fonte.
- Il sistema deve fornire almeno un sistema per potersi collegare alla fonte originale

R9: Identificazione del social.

- Il sistema deve marcare ogni post con il socialnetwork da cui arrivano i dato.

R10: Interfaccia ricerca

- Fornire un'interfaccia adeguato a sviluppatori terzi per proporre nuove implementazioni di sistemi di ricerca verso i socialnetworks
- le varie implementazioni devono essere indipendenti dal sistema
- le varie implementazioni devono essere completamente isolate tra loro

Strutture dati

La comunicazione tra client e il servizio SocialHub deve avvenire attraverso l'uso di oggetti Json ben definito sia per messaggi rivolti al servizio sia quelli che partono verso il client.

Utente:

Json semplificato di quello fornito da Twitter con presente solamente

- ID twitter
- nome utente
- foto utente

Risultato ricerca:

lista Json composto da Post. I post ottenuti dalla ricerca dei risultati vengono suddivisi in finestre (per semplicità in questo progetto vengono suddivisi in 5 post per finestra per ciascun servizio richiesto) che il client può navigare esclusivamente in una sola direzione. Sarà cura del client mantenere eventuali dati precedenti in quanto il server non si impegna nel mantenere la cache delle finestre già richieste. Il motivo di questa scelta è dato dal fatto che non si conosce la natura di come viene gestito da ciascun singolo socialnetwork e quindi si garantisce solamente il modello di iterazione più semplice.

Post:

Json basilare che rappresenta il post contenente:

- Titolo, sommario del post
- Contenuto, opzionale in quanto le foto recuperato da Flickr possono non avere una descrizione.
- thumbnail, opzionale contenente un URI per individuare l'eventuale miniatura associata al post, nel caso di Flickr rappresenta il vero contenuto dell'informazione trovato
- src, URI associata alla fonte del post
- servizio, nome del social network associato

History:

lista Json contente le richieste di ricerca già effettuate da un utente registrato.

History entry:

Oggetto Json che rappresenta il risultato di una ricerca di un utente composto da:

- idHistory: id univoco che individua il record

- parole: lista di parole chiavi ricercate
- servizi: lista di servizi su si ha effettuato la ricerca
- data: data in cui è stato effettuata la richiesta rappresentato dal timestamp unix.

Protocollo per le richieste

Il sistema utilizzerà un protocollo simil REST che mantiene la sessione corrente.

GET,POST /login/web:

non richiede nessun tipo di parametro in quanto si reindirizza direttamente alla richiesta Twitter. al termine si viene reindirizzati alla home

GET,POST /login/web/reception:

Richiesta di callback per Twitter, questa richiesta termina /login/web.

GET,POST /login/app:

- senza parametri restituisce l'URI per ottenere poter effettuare il login via Twitter
- richiede l'attivazione obbligatoria della sessione in quanto memorizza il token della richiesta per Twitter da usare per il completamento della richiesta

GET,POST /login/app/reception

- Richiesta di callback per Twitter. Non necessita di essere richiamata dalla stessa sessione di /login/app in quanto memorizza l'utente nell'application context utilizzando il token per la richiesta Twitter.

GET,POST /utente/app/init

Termina l'azione /login/app, deve essere richiesta nella stessa sessione di /login/app in quanto recupera il token precedentemente memorizzato nella sessione per poter recuperare l'utente salvato da /login/app/reception.

GET,POST /logout :

non richiede nessun tipo di parametro e viene condiviso sia dalla chiamata per applicazione web che per applicazioni.

- L'effetto ottenuto è la cancellazione dell'utente associato alla sessione attuale
- l'invalidazione della sessione attuale
- re-indirizzamento alla root

GET /search:

parametri richiesti: q={tags: String[], services: String[]}

- effettua la ricerca dei post usando le keys presenti in q dei servizi specificati in services, nel caso services non sia presente verrano usati tutti i servizi presenti.

- Restituisce un Json con la lista dei risultati
- la query usata verrà memorizzata nella history.

GET /history:

nessun parametro richiesto.
Restituisce un Json contenente tutta history dell'utente oppure errore 404 con il json {errors: true, message: "Non sei loggato"}

GET /utente:

nessun parametro richiesto.
Restituisce il json rappresentante l'utente nella sessione corrente oppure errore 404 con il json {errors: true, message: "Non sei loggato"}

Implementazione

Il progetto si concentra principalmente nella creazione del servizio in quanto i client sono completamente indipendenti, e chiunque rispetti il protocollo di comunicazione sopra descritto può utilizzarlo. Il server implementa una versione personale del client attraverso l'uso del di Angularjs 2.

Il servizio

Il servizio è composto principalmente da 3 moduli:

- Il modello dei dati usato per la memorizzazione della cronologia dell'utente, e degli ID dei utenti registrati
- Interfaccia servizio-socialnetworks rappresentato dal package dell'interfaccia per uniformare il risultato di ricerca ottenuto dalle single implementazioni che sono delegati a gestire le modalità di connessione e le differenze dei protocolli e sistemi di ricerca individuali.
- Interfaccia servizio-clients rappresentato dai servlets che implementato il protocollo di comunicazione effettuando le ricerche utilizzando le implementazioni sopra descritto attraverso l'interfaccia.

Il modello:

Per l'implementazione del modello si è scelto l'uso di classi enumeratori per la rappresentazione delle tabelle in quanto garantiscono un'ottima stabilità e la praticità di poter usare i valori degli enumeratori anche come stringhe o comunque ottenerne il valore stringa semplicemente con la funzione name.
Per avere una struttura più omogenea è stata definita l'interfaccia DBTableInterface per raccogliere gli enumeratorori tabelle semplificandone l'uso.
Per semplificare la rappresentazione dei tipi di dati e le restrizioni dei database sono stati creati 2 package apposi per tale scopo.
Un'istanza del tipo di dato opportuno viene istanzziato alla creazione del valore dell'enumeratore e una costanti per la rappresentare le restrizioni.

Interfaccia servizio-client:

L'interfaccia con i client che vogliono utilizzare i servizi offerti da SocialHub devo attenersi al protocollo di comunicazione sopra descritto che viene garantito da servlet realizzano l'implementazione per il protocollo.
Nel caso del sistema di login vengono utilizzati diversi servlet per non concentrare tutto il lavoro in un' unico servlet parametrizzato.
Tutti i servlet condividono un oggetto properties contenenti i parameti base vine istanziato all'interto dell'application context in modo che non sia necessario doverlo ricostruire ogni volta in quanto uguale per tutta l'applicazione.
Ogni volta che un utente si connette al sistema viene creato una sessione a lui dedicato, anche se non dovesse essere loggato che viene usato per condividere informazioni con altri servlet all'interno della stessa sessione, come ad esempio l'utente.
Il servlet che gestisce la ricerca invece ha anche un'istanza di classe che mantiene tutte le classi dei SocialNetworks, abbinato ai parametri corrispondenti, disponibili istanziati con l'attivazione del servlet stesso non modo che sia disponibile immediatamente. Le ricerche invece vengono effettuate da oggetti appositamente istanziati con le classi del provider per ciascuna sessione che li memorizza per usi futuri e mantenimento dello stato della ricerca per richiedere ulteriori informazioni.
La history invece viene recuperato ogni volta dal database per garantire la sincronizzazione dello stesso utente che ha effettuato l'accesso utilizzando piattaforme differenti. Il sistema manda al più 100 records per richiesta.
SocialHumb implementa 2 differenti sistema di login con logiche simili basati su OAuth di Twitter.
Il primo risponde alle richiede dell'implementazione del cliente interno utilizzando i /login/web che ha solo 2 passaggi

1. Richiesta al server per effettuare il login, il server reindirizza il browser alla pagina login di Twitter
2. Il sistema di login Twitter reindirizza su /login/web/reception che memorizza nella sessione l'utente utilizzando i parametri ritornati da Twitter

Il sistema per applicazioni di terze parti invece risulta più complessa nei passaggi anche se il procedimento è praticamente identico ed è composto da:

1. Richiesta al server per effettuare il login a /login/app che restituisce un oggetto contenente il token di Twitter e la pagina Twitter per effettuare il login. Il server memorizza nella sessione corrente il token per recuperare l'utente che verrà generato nei passi successivi.
2. In questo caso il re-indirizzamento deve essere gestito dall'applicazione client e non ha necessità di usare la stessa sessione su cui ha effettuato l'accesso.
3. La pagina di login di Twitter del passo 2 richiama il callback preparato su /login/app/reception che memorizza nel contesto globale dell'applicazione utilizzando una chiave composta anche dal token Twitter(per semplicità rappresenta semplicemente il suffisso della chiave utilizzata per memorizzare l'utente nella sessione attuale) che visualizza un messaggio di successo nel caso la procedura sia andato a buon fine.
4. L'applicazione per terminare la procedura di login deve confermare utilizzando /utente/init/app che recupera il token Twitter memorizzato nel passo 1 che utilizza per recuperare l'utente creato nel passo 3.

Nonostante la complessità e il numero dei passaggi il sistema garantisce un accesso sicuro senza scambiare parametri segreti con il client in modo esplicito ma sfrutta il sistema di login consolidato di Twitter.

Interfaccia servizio-socialnetworks

L'attenzione principale per questo modulo è rivolto principalmente alla costruzione di un'interfaccia semplice, solida e duratura nel tempo che riesca a rimanere invariato più a lungo possibile delegando tutti i cambiamente di implementazione dei singoli socials alle implementazioni che devono essere implementati in modo completamente autonomi.
L'interfaccia è composta principalmente dai JavaBean che rappresentano direttamente i dati richiesti dalla comunicazione con il client e dall'interfaccia SocialNetwork che rappresenta l'interfaccia base richiesto alle singole istanze che dovranno eseguire il lavoro effettivo.

- HistoryBean: rappresenta la cronologia dell'utente composto da una lista di HistoryEntryBean che rappresenta un singolo record della cronologia
- RisultatiBean: rappresenta la lista dei risultati ottenuti dall'interrogazione dei socialnetworks composto da PostBean che rappresenta un singolo post.
- UtenteBean: rappresenta l'utente da ritorna alla richiesta.

Anche se i JavaBean sono ben noti e largamente utilizzati sono state create queste interfacce per garantire comunque che eventuali applicazioni implementi correttamente questi Bean che saranno utilizzati da SocialHub. La sola convenzione potrebbe bastare ma non offre certezze assolute e si è preferito comunque lasciare liberata ai programmatori per l'implementazione finale.
Il sistema comunque ha implementato delle classi base per poter essere operativo da subito.

Interfaccia SocialNetworks

è la parte fondamentale per il buon funzionamento del sistema e garantire longevità del sistema ed è per questo che si è cercato di mantenerlo il più semplice possibile e delegare alle implementazioni finale il resto delle complicazioni, quello che deve garantire sono:

- gestire la connessione con il servizio specifico
- effettuare la ricerca
- permettere di ottenere maggiori risultati sulla ricerca simile agli iterratori
- settare il numero di risultati da restituire per richiesta

Eventuali parametri dovranno essere memorizzati all'interno di un file xml all'interno della cartella WEB-INF/sn_services insieme al full-class name e al nome del servizio. Si è scelti di utilizzare il file xml con il formato dei Properties di java per semplificare la definizione del fine, e personalmente trovo più espressivo l'uso di xml e di conseguenza più facile da capire e interpretare.
Il Servlet che gestisce le richieste per la ricerca con i client effettuera una ricerca dei fine presenti dentro WEB-INF/ns_services per creare delle istanze SocialService che memorizza il file delle proprietà associato alla loro classe e può effettuale le operazioni basi per l'istanziazione di un socialnetwork pronto per effettuare le ricerche e memorizzate all'interno di un oggetto ServiceProvider che è hash map per gestire velocemente la ricerca dei SocialService utilizzando il nome del servizio.

Il client, implementato da SocialHub

L'implementazione di questo primo client è puramente dimostrativo e più orienta all'esplorazione delle potenzialità dei brower moderni utilizzando opportuni Framework di sviluppo javascript, in questo caso si è scelto AngularJs2, e non affronta la parte grafica e la strutturazione della parte interazione utente.

Perché AngularJs2?

AngularJs è un Framework che permette di creare siti web utilizzando il pattern Model-View-Controller orientato alle componenti e servizi dove ogni componente rappresenta la parte Controller, le view sono rappresentati da template Html che però usa un linguaggio di templating molto potente che possono rendere il pattern più simile a una GUI piutosto che a un MVC.
I servizi rappresentano i sistemi di comunicazioni tra le componenti e sistemi esterni e rappresentano un'altra parte del controller. Il modello generalmente è rappresentato da oggetti javascript che vengono gestiti dai Controller e dai Servizi.
Oltre a Angular ci sono anche altri ottimi concorrenti, ma personalmente ho trovato molto appropriato le scelte che hanno fatto il team di sviluppo con la release della seconda versione del Framework con scelte molto azzardate e radicati.
Primo fra tutte è l'uso di Typescript che è un superset di Ecmascript 6 e compilabile in javascript. Questo nuovo linguaggio sviluppato da Microsoft aggiunge i tipi all'interno di javascript in modo più preciso e un linguaggio molti più appropriato per uno sviluppo OOP. Di fatto eventuali conflitto di tipi non comporta al fallimento della compilazione ma crea dei warning che aiutano a debbuggare e commettere meno errori causati da errori dovuto alla cattiva gestione delle variabili non tipati. Un altro vantaggio che questo linguaggio ha rispetto ad altri suoi simili è il fatto che per riutilizzare librerie preesistenti scritti in javascript è sufficiente creare un file di definizione che crea il link tra il compilatore typescprt e la libreria senza doverlo riscrivere completamente.
L'altra scelta principale è la sostituzione di Reactivejs alle promise che aumentano la reattività del databinding e delle richieste http asincrone effettuati. Anche la libreare Reactive è stata creato da Microsoft e lo scopo principale di questa libreria è la gestione di flussi di dati come se fossero array, utilizzando il pattern pablish/subscribe, permettendo un'ottima gestione con potenti rimappature, filtri e altre funzioni che non ho ancora affrontato come avrei dovuto. L'esempio migliore prodotto è la histori.service.
Nella nuova versione è stati semplificata la gestione del progetto attraverso i decoratori che rendono il codice molto più leggibile e gestibile.
Il sistema di associazione dei template permette di scrivere direttamente in una variabile del decoratore del componente oppure utilizzare un file esterno, e oltre alla struttura del componente viene dato anche la possibilità di applicare stili isolati al componente gestiti direttamente da Angular specificando nel decoratore lo stile oppure uno o più fogli di stili esterni. Credo che questo ultimo punto sia stato il motivo principale per la scelta del Framework in quanto apprezzo particolarmente la separazione netta tra il codice e delle viste che ho apprezzato anche su Android.

Implementazione

Seguendo la filosofia Angular il progetto è stato sviluppato in componente e servizi ragruppati per rappresentare un'entità abinato anche a un template ed eventuali fogli di stili che ho inserito in apposite cartelle separate dal codice.
Seguendo le orme della guida introduttiva di Angular il progetto è stato sviluppato utilizzando le librerie fornite dalla npm che ha avuto funzione di dependency resolver e l'intermediario per la compilazione automatica di typescript.
Il sistema di gestione dei package/moduli javascript all'interno del progetto è stato mantenuto Systemjs illustrato nella guida, il compilatore di typescript si occupa completamente della creazione dei moduli e risolvere in seguito i importo nel codice compilato.
Il codice sorgente è stato sviluppato dentro ad app_src mentre il compilato posto all'interno di app.

Struttura

nella root sono presenti il file main che si occupata di gestire il boostrap di tutto il progetto e il componente radice dove risiedono poi tutti gli altri elementi app.component

Configs

qui sono presenti tutti parametri di configurazione, attualmente è presente solo il modulo contenente le actions che corrispondono agli indirizzi per le richieste al servizio

History

Modulo per la gestione della cronologia composto da:

- 2 componenti che rappresentano rispettivamente la lista dei record e i singoli records.
- 1 servizio per la gestione della cronologia. Viene usato dal componente history per il recupero e aggiornamento della lista della cronologia. Viene usato anche dal componente post-result per aggiornare la cronologia a seguito di una ricerca con esito positivo. Utilizza il pattern publish/subscrib per aggiornare la history.
- 2 modelli per rappresentare la cronologia e i suoi records del tutto equivalenti a presenti sul server, solo privati dei tutti i setter come anche gli altri modelli utilizzati nel progetto.

post

Modulo principale del progetto composto da:

- 3 componenti che rappresentano la lista dei post e i singoli post con i più un component posts-result che in realtà è nato più per un errore durante lo sviluppo ed è completamente ridondante rispetto al componente principale app.
- un servizio per effettuare la ricerca sempre utilizzando il pattern p/s attraverso Reactive ma in modo diverso da history in quanto questo non sfrutta a pieno le potenzialità offerte da Reactive in quanto sviluppato prima.
- 1 modello che rappresenta un singolo poste
- 1 oggetto per rappresentare una query con abbinato un builder per semplificarne la costruzione

Utente

Modulo per la gestione dell'utente composto da:

- 1 componente per rappresentare l'utente
- 1 servizio per la gestione del login e il recupero delle informazione sull'utente
- 1 modello che rappresenta l'utente

Socialnetwork/Token

Questi 2 moduli sono nati per semplificare la visualizzazione e la gestione interattiva dell'interfaccia utente per effettuare la richiesta di ricerca dell'utente.