UNIVESITÁ DEGLI STUDI DI MILANO
LAUREA TRIENNALE IN COMUNICAZIONE DIGITALE
PROGETTO LABORATORIO DI RETI DI CALCOLATORI

Master Quiz

Matteo Zignani

18 agosto 2014

1 PRESENTAZIONE DEL PROBLEMA

Lo studente deve sviluppare un gioco-quiz che sfrutti la social network fra gli utenti registrati al servizio. In particolare ogni utente deve rispondere al maggior numero di domande generate ogni due minuti con la possibilità di chiedere la risposta giusta ai propri amici.
Il servizio che lo studente deve implementare si basa su un'architettura client/server. Il server deve gestire la rete sociale tra gli utenti iscritti, offrendo anche la registrazione di nuovi utenti, e offrire dei servizi per lo svolgimento del gioco. A tal fine, il server mette a disposizione diverse risorse richiedibili utilizzando il protocollo HTTP che permettono la gestione, la richiesta e la pubblicazione della richiesta stessa , secondo i vincoli specificati in seguito. Per agevolare la pubblicazione dei post su Facebook è stato creato un utente fittizio di nome Giovanni Algoritmo sul cui profilo è possibile postare gli stati.
Lo studente deve sviluppare solamente la parte server dell'architettura descritta. Per quanto riguarda il client si possono utilizzare per la fase di test: telnet (Linux e Mac), putty (Windows). Le funzionalità verranno testate dal docente con questi due strumenti. Il server DEVE essere MULTITHREAD.

2 API

Nelle seguenti sottosezioni vengono descritti le componenti del server e le risorse che esso mette a disposizione. **Gli esempi presentati di seguito sono da considerarsi come tali, pertanto nella costruzione delle richieste e delle risposte HTTP ci si deve rifare alla sintassi HTTP presentata sul libro di testo.**

2.1 FORMATO DELLA RICHIESTA

Il server riconosce come valide le richieste che seguono il protocollo HTTP. In particolare il server riconosce solo richieste con metodo di accesso di tipo GET. Per questo motivo lo studente NON deve implementare un server HTTP completo ma un suo piccolo sottoinsieme. Un esempio di possibile richiesta valida può essere:

```
GET <URL Risorsa con eventuali parametri> HTTP/1.0
user-agent: Mozilla 5.0
host: localhost:6000
content-type: text/html
<rigaVuota>
```

Il server deve verificare la validità della riga di intestazione e ignorare tutti i campi settati nella definizione dell'header. Una volta rilevata la stringa vuota di terminazione della richiesta, il server deve eseguire l'operazione richiesta dall'URL e rispondere secondo il formato HTTP. Si noti che l'URL della riga di intestazione della richiesta deve seguire la sintassi specifica di un URL (vedi slide corso). Se la richiesta è andata a buon fine, il formato della risposta è:

```
HTTP/1.0 200 OK
<rigaVuota>
<messaggio>
```

dove < $messaggio$ > è un messaggio testuale che dipende dalla risorsa invocata. In caso di errore il formato della risposta sarà:

```
HTTP/1.0 400 ERRORE
<rigaVuota>
<messaggio>
```

Anche in questo caso < $messaggio$ > contiene un messaggio di errore dipendente dalla risorsa invocata.
Vincoli imposti sulle richieste:

- **Tutte le seguenti risorse, parametri e valori sono case-sensitive.**
- **L'ordine dei parametri nella richiesta non ha importanza**

2.2 CREAZIONE DI UN NUOVO UTENTE

Al momento dell'avvio il server non contiene alcun utente. Per inserire un nuovo utente, il server mette a disposizione la risorsa $aggiungiUtente$. Tale risorsa accetta come unico parametro valido $nomeUtente$, che indica il nome dell'utente che si vuole inserire. Un possibile esempio di richiesta valida è:

```
GET /aggiungiUtente?nomeUtente=mario HTTP/1.0
user-agent: Mozilla 5.0
host: localhost:6000
content-type: text/html
<rigaVuota>
```

Una possibile risposta positiva è data da:

```
HTTP/1.0 200 OK

Utente mario inserito con successo. Browser: Mozilla 5.0
```

Possibili errori da gestire sono l'inserimento di un utente già esistente.

2.3 AGGIUNGERE UNA RELAZIONE DI AMICIZIA

Un utente può aggiungere un amico alla sua cerchia utilizzando la risorsa $aggiungiAmico$. La risorsa *aggiungiAmico* accetta come unici parametri validi $nomeUtente$ e $nomeAmico$, il primo indica l'utente che vuole creare l'amicizia, mentre il secondo indica il nome (stringa) dell'utente verso cui creare una relazione di amicizia. L'arco che viene creato è non orientato e non necessita che l'utente destinatario della richiesta accetti la creazione. Un possibile esempio di richiesta valida è il seguente:

```
GET /aggiungiAmico?nomeAmico=ambrogio&nomeUtente=mario HTTP/1.0
user-agent: Mozilla 5.0
host: localhost:6000
content-type: text/html
<rigaVuota>
```

Una possibile risposta positiva è data da:

```
HTTP/1.0 200 OK

Amicizia con ambrogio creata con successo
```

Possibili errori da gestire sono l'assenza del nodo di destinazione/sorgente della richiesta o la richiesta di creazione di una relazione che già esiste.

2.4 LISTA DEGLI AMICI

Un utente può visualizzare i propri amici utilizzando la risorsa $vediAmici$. La risorsa necessita dell'argomento $nomeUtente$: l'utente che richiede la lista degli amici. Nel corpo della risposta, $vediAmici$ restituisce un array JSON contenente una serie di oggetti JSON che rappresentano gli amici. Ogni oggetto JSON è costituito da un campo: $idUtente$ contenente il nome utente (stringa) dell'amico. Un possibile esempio di richiesta valida è il seguente:

```
GET /vediAmici?nomeUtente=mario HTTP/1.0
user-agent: Mozilla 5.0
host: localhost:6000
content-type: text/html
<rigaVuota>
```

Una possibile risposta positiva è data da:

```
HTTP/1.0 200 OK

[
{"idUtente": "ambrogio"},
{"idUtente": "michele"},
{"idUtente": "marco"}
]
```

Nell'esempio michele, amico di mario, si è registrato con il browser Mozilla 5.0. Possibili errori sono costituiti dall'errata sintassi della risorsa oppure da un nome utente non esistente.
Un secondo insieme delle risorse disponibili riguarda la gestione del quiz e l'interazione con gli amici per poter rispondere ad una domanda. Master Quiz implementa una modalità di quiz social in cui l'utente che affronta una domanda può chiedere la risposta ad uno dei suoi amici entro un prefissato tempo.
Le domande vengono generate e gestite in modo automatico dal server, non esistono amministratori che inseriscono manualmente domande o che decidono un eventuale scadenza anticipata della stessa. Il server estrae in modo casuale una domanda da una lista di domande/risposte memorizzate su un file e determina la data di scadenza. Le domande sono create in modo sequenziale e distanti una dall'altra **2 minuti** e valgono per tutti gli utenti iscritti. Ogni due minuti, il server quindi somministra un nuovo quiz con data di scadenza $d_s = d_c + \Delta t$ dove d_c è la data di creazione e Δt è un intero (numero di minuti) che viene passato come argomento alla classe che modella il server. Per esempio se il server viene creato alle ore 9:00 mediante

```
java server 3
```

il primo quiz viene creato alle ore 9:02 e termina alle ore 9:05, il secondo inizierà alle ore 9:04 e terminerà alle ore 9:07, etc ...
Ad ogni domanda viene assegnato un identificatore univoco tra le domande disponibili, ciò significa che è possibile identificare in modo univoco qualsiasi domanda non scaduta. Una volta terminato il quiz è possibile riusare l'id utilizzato verificandone dell'univocità. Il server deve tenere traccia dei quiz validi ed chiudere i quiz che sono terminati assegnando un punto a tutti coloro che hanno risposto correttamente.

2.4.1 Richiesta risposta ad un singolo amico

La risorsa $richiediRisposta$ permette ad un utente di richiedere una risposta di una specifica domanda ad un solo amico. La risorsa richiede tre parametri: $idDomanda$ specifica la domanda, $nomeAmico$ specifica l'utente a cui chiedere la risposta e $idUtente$ è l'utente che chiede la risposta all'amico. **Non è possibile richiedere una risposta di una stessa domanda a più amici e non è possibile richiedere una risposta ad utenti che non sono amici del richiedente.** Un possibile esempio di richiesta valida è il seguente:

```
GET /richiediRisposta?nomeUtente=mario&nomeAmico=michele&idDomanda=3 HTTP/1.0
host: localhost:8000
content-type: text/html
```

```
<rigaVuota>
```

Una possibile risposta positiva è data da:

```
HTTP/1.0 200 OK

Richiesta inoltrata con successo
```

Possibili errori possono essere causati dalla mancata presenza degli utenti oppure da un $idDomanda$ non valido.

2.4.2 VISUALIZZA RICHIESTE DI RISPOSTA

La risorsa $vediRichieste$ permette ad un utente di vedere tutte le richieste che gli sono state inoltrate. La risorsa prevede un solo parametro denominato $idUtente$, indicante l'utente che vuole interrogare la risorsa. $vediRichieste$ restituisce un array JSON contenente una serie di oggetti JSON che rappresentano gli amici. Ogni oggetto JSON è costituito da due campi: $idUtente$ contenente il nome utente (stringa) che ha inviato una richiesta e $idDomanda$, id della domanda oggetto della richiesta di aiuto. Un possibile esempio di richiesta valida è il seguente:

```
GET /vediRichieste?idUtente=mario HTTP/1.0
host: localhost:6000
content-type: text/html
<rigaVuota>
```

Una possibile risposta positiva è data da:

```
HTTP/1.0 200 OK

[
{"idUtente": "ambrogio", "idDomanda": 3},
{"idUtente": "michele", "idDomanda": 4}
]
```

Possibili errori possono essere dovuti all'assenza di $idUtente$ nella rete sociale.

2.4.3 QUIZ DISPONIBILI

Un utente può verificare le domande ancora valide attraverso la risorsa $getQuiz$. La risorsa non prevede alcun parametro e restituisce un array JSON di oggetti JSON. Ogni oggetto contiene due campi: $idDomanda$ contenente l'identificatore univoco della domanda e $testoDomanda$ che contiene il testo della domanda. Un possibile esempio di richiesta valida è il seguente:

```
GET /getQuiz HTTP/1.0
host: localhost:6000
content-type: text/html
<rigaVuota>
```

Una possibile risposta positiva è data da:

```
HTTP/1.0 200 OK

[
{"testoDomanda": "Chi ha vinto il mondiale in Brasile?", "idDomanda": 3},
{"testoDomanda": "Quanto è lungo il fiume Po?", "idDomanda": 4}
]
```

2.4.4 RISPONDI

Un utente può risolvere un quiz mediante la risorsa $rispondiDomanda$. La risorsa riceve tre parametri: $idDomanda$, $risposta$ e $idUtente$; il primo rappresenta l'id del quiz, il secondo la risposta e il terzo parametro indica l'utente che fornisce la risposta. E' possibile cambiare una risposta sino al termine del quiz invocando più volte la risorsa $rispondi$ sullo stesso id. Allo scadere del quiz, se la risposta è corretta viene assegnato un punto all'utente. Un possibile esempio di richiesta valida è il seguente:

```
GET /rispondiDomanda?idUtente=mario&idDomanda=3&risposta=germania HTTP/1.0
host: localhost:8000
content-type: text/html
<rigaVuota>
```

Una possibile risposta positiva è data da:

```
HTTP/1.0 200 OK

Risposta ricevuta
```

2.4.5 PUNTEGGIO ATTUALE

Un utente può richiedere il punteggio attuale attraverso la risorsa $getPunteggio$. La risorsa richiede solo $idUtente$ come parametro necessario. La risorsa pubblica un messaggio sul profilo di un utente definito dall'id **100007294060146**. Il messaggio ha la forma 'Punteggio $< idUtente >$: $< punteggio >$'. Un possibile esempio di richiesta valida è il seguente:

```
GET /getPunteggio?idUtente=mario HTTP/1.0
host: localhost:8000
content-type: text/html
<rigaVuota>
```

La risorsa, oltre alla pubblicazione sul profilo Facebook (vedi figura 2.1), deve inviare anche una risposta HTTP che comunica l'avvenuta pubblicazione. Per esempio:

```
HTTP/1.0 200 OK

Pubblicazione del punteggio avvenuta con successo
```

Figura 2.1: Esempio di pubblicazione.

L'access token da utilizzare per la pubblicazione è il seguente:
CAADaUwpF1ZAEBAAVZCFkokQdiruOrt6dvDp5jrGd6CFb
s41eUgnJzLU9JdrTH2CUieczU7QnzfFoZC0FByrZAjMAQUfUbgq5X
jijqxgbfvprWt90GwYsKisWRlZCZBZCTfZA6P8bvyLVkOcne0UnxQl
uqhwgtzBXRjfalsZB6Yhzbw6tORnqcMMcMUnz8jjMG73RZBq1GOZBsAPUVovyf1Rv6i9

3 Modalità di consegna

Il progetto è individuale e verranno presi provvedimenti nel caso ci sia il sospetto di una eventuale copiatura. In particolare non verranno ammessi alla discussione del progetto e quindi saranno automaticamente esclusi dall'appello quei progetti con almeno un file con indice di similarità superiore al 70% (moss, tool di Stanford). Questo per non incentivare progetti scritti singolarmente ma frutto di brain-storming troppo di gruppo.
Il linguaggio da utilizzare per lo sviluppo è Java. Il codice del progetto e i relativi jar delle librerie utilizzate devono essere inviati al docente in una directory compressa (zip o tar.gz) denominata 'progettoSettembre2014'. La mail deve avere come oggetto 'ProgettoSettembre2014' e contenere nel corpo nome, cognome e numero di matricola dello studente. Il materiale deve essere consegnato entro le **16:59:59** del **18 settembre 2014**. Valgono tassativamente le seguenti regole:

- Il codice che non compila non verrà considerato
- I progetti consegnati dopo la data di consegna non saranno ritenuti validi
- Eventuali errori sintattici riscontrati nell'implementazione del protocollo HTTP oppure nel formato JSON rendono il progetto insufficiente.
- Per essere considerato sufficiente e quindi ammissibile per la discussione, il progetto deve implementare correttamente tutte le funzioni precedentemente illustrate.

Principali criteri di valutazione del codice:

- Gestione di possibili errori nell'input da parte dell'utente
- Modularità del codice e rispetto del paradigma ad oggetti
- Gestione oculata delle eccezioni

- Uso appropriato dei commenti
- Gestione e risoluzione di eventuali problematiche legate alla concorrenza sia lato client sia lato server

Per chiarimenti, dubbi potete contattare il docente all'indirizzo matteo.zignani@unimi.it, specificando come oggetto obbligatorio 'ProgettoReti'. Eventuali modifiche al testo del progetto saranno comunicate sulla pagina web del corso.