Implementazione di un sistema di Aste

Gabriele Gemmi
gabriele.gemmi@stud.unifi.it

Febbraio 2017

Introduzione

Contents

1 Protocollo 2
1.1 Messaggio di risposta . . . 3
1.2 Autenticazione . . . 3
1.2.1 Sessioni . . . 3
1.2.2 Formato . . . 4
1.3 Gestione delle Notifiche . . . 4
1.3.1 Registrazione . . . 5
1.3.2 Messaggi di notifica . . . 5
1.4 Categorie . . . 5
1.4.1 Registrazione categoria . . . 6
1.4.2 Lista delle categorie . . . 6
1.4.3 Ricerca di una categoria . . . 6
1.5 Prodotti . . . 7
1.5.1 Registra un nuovo prodotto . . . 7
1.5.2 Ricerca di un prodotto . . . 7
1.5.3 Lista tutti i prodotti . . . 8
1.5.4 Messaggio di risposta . . . 8
1.6 Asta . . . 8
1.6.1 Offerta . . . 8
1.6.2 Ritiro di un offerta . . . 9
1.6.3 Chiusura di un'asta . . . 9
1.6.4 Messaggio di risposta . . . 9

2 Implementazione 10
2.1 Server . . . 10
2.1.1 IbaiServer . . . 10

2.1.2 ClientManager 10
2.1.3 model 10
2.2 Client 11
2.2.1 IbaiClient 11
2.3 Librerie utilizzate 11

3 Tests 11

1 Protocollo

Per sviluppare questo progetto era necessario definire un protocollo per la comunicazione client-server. Per garantire la scalabilità del sistema è stato scelto un protocollo stateless, nonchè connectionless.
E' stato utlizzato JSON come formato per lo scambio di dati in quanto erano disponibili nella libreria standard python i metodi per serializzare e deserializzare gli oggetti.

Il formato di un generico messaggio è:

```
{
  "msg_id": <id>
}
```

La prima cifra del campo msg_id indica il tipo di richiesta, la seconda indica il metodo

- -1: Messaggio di risposta a un comando
- 1: Autenticazione
  - 11: Log in
  - 12: Log out
- 2: Notifiche
  - 21: Registra l'endpoint per le notifiche
  - 22: Messaggio di notifica
- 3: Categorie
  - 31: Registra nuova categoria
  - 32: Ricerca di una categoria
  - 33: Lista tutte le categorie
- 4: Prodotti
  - 41: Registra un nuovo prodotto
  - 42: Ricerca un prodotto
  - 43: Lista tutti i prodotti
- 5: Asta
  - 51: Fai un offerta
  - 52: Cancella la tua ultima offerta
  - 53: Chiudi l'asta

1.1 Messaggio di risposta

Il formato di un generico messaggio di risposta è il seguente.
Se non è esplicitamente indicato un'altro formato per la risposta, allora la risposta sarà di questo formato.

```
{
  "msg_id": -1
  "response" : <code>
}
```

Campo	Tipo	Descrizione
response	int	'1' in caso di successo '-1' comando non sia valido '0' errore '2' sessione scaduta '3' sessione non valida

Questo messaggio può essere esteso dalle varie funzioni aggiungendo campi e codici di risposta.

1.2 Autenticazione

La prima cosa che deve fare un client quando si connette al server è autenticarsi utilizzando un username ed una password. Il server mantiene una lista di utenti con relative password, quando il client prova ad autenticarsi vengono confrontati e se sono corretti l'autenticazione è avvenuta. Per evitare che nel server vengano memorizzate le password in chiaro, è stato scelto di memorizzare e trasmettere l'hash md5 della password. Il Client si occuperà di eseguire l'hash md5 della password inserita dall'utente e di inviarlo al server, che lo confronterà con quello memorizzato.

1.2.1 Sessioni

Per gestire le sessioni mantenendo un procollo stateless è stato scelto di generare un token ed inviarlo al client. Una volta effettuato il login il server restituirà un token al client che verrà utilizzato per autenticare tutti i comandi.
Il token inviato al client è della forma:

```
{
  "username" : <user>
  "expiration": <expiration>
  "signature": <signed proof of autenticity>
}
```

Campo	Tipo	Descrizione
username	stringa	Username dell'utente
expiration	int	Data e ora di scadenza della sessione (in secondi)
signature	stringa	Codice di verifica del messaggio

Il campo signature è un hash dei precedenti due campi (username e expiration) più una chiave segreta generata dal server. In questa maniera il server potrà verificare che il token sia autentico senza doverne mantenere una copia in memoria.

1.2.2 Formato

Il messaggio di autenticazione è del formato:

```
{
  "msg_id": 11,
  "user": <username>,
  "pass": <hash of pwd>
}
```

Campo	Tipo	Descrizione
user	stringa	Username dell'utente
pass	stringa	Hash MD5 della password

La risposta è del formato:

```
{
  "msg_id": -1,
  "response": <code>
  "token": <token>
}
```

Campo	Tipo	Descrizione
response	int	codice di risposta
token	json	token per la sessione

1.3 Gestione delle Notifiche

Il server di aste può inviare delle notifiche asincrone ai client per comunicare aggiornamenti di stato. Per esempio il client riceverà una notifica quando un'asta viene terminata o la sua offerta viene superata.

1.3.1 Registrazione

Per ricevere delle notifiche asincrone il client dovrà stare in ascolto su una determinata porta, e dovrà comunicare il proprio indirizzo e la porta al server. Per fare questo è disponibile un comando. Di seguito il formato:

```
{
  "token": <token>,
  "msg_id": 21,
  "host": <hostname>,
  "port": <port>
}
```

Campo	Tipo	Descrizione
token	json	Token per la sessione
host	string	Hostname del client
port	int	Porta alla quale il client riceve le notifiche

1.3.2 Messaggi di notifica

Un messaggio di notiica, inviato dal server al client, rispetta il formato:

```
{
  "msg_id": 22,
  "code": <notification code>,
  "text": <human readable meaning>
}
```

Campo	Tipo	Descrizione
code	int	codice di notifica -1 Notifiche OK 1 Asta chiusa 2 Hai vinto
text	string	Messaggio di notifica

1.4 Categorie

I prodotti in vendita nel sistema di aste sono suddivisi per categorie merceologiche. E' possibile effettuare varie operazioni sulle cateogorie: Ogni categoria è identificata in maniera univoca dal suo nome.

1.4.1 Registrazione categoria

Attraverso il seguente comando è possibile aggiungere una nuova categoria
Il nome della categoria sarà utilizzato per identificarla.

```
{
  "token": <token>,
  "msg_id": 31,
  "category": <category name>
}
```

Campo	Tipo	Descrizione
category	string	Nome della categoria, univoco.

1.4.2 Lista delle categorie

Attraverso il seguente comando è possibile ottenere una lista delle categorie

```
{
  "token": <token>,
  "msg_id": 33
}
```

Il formato della risposta è:

```
{
  "msg_id": -1,
  "response": <code>
  "categories":
  [
    <category1>,
    <category2>
  ]
}
```

1.4.3 Ricerca di una categoria

Attraverso il seguente comando è possibile ricercare tra le categorie per nome

```
{
  "token": <token>,
  "msg_id": 32,
  "category": <category name>
}
```

Campo	Tipo	Descrizione
category	string	Espressione da ricercare.

Il formato della risposta è lo stesso del precedente comando.

1.5 Prodotti

All'interno delle categorie merceologiche sono presenti i prodotti. E' possibile aggiungere nuovi prodotti, ricercare i prodotti per nome o visualizzarli tutti.

1.5.1 Registra un nuovo prodotto

Il seguente comando aggiunge una nuova asta per il determinato prodotto.

```
{
  "token": <token>,
  "msg_id": 41,
  "category": <cat_name>,
  "product": <prod_name>,
  "price": <price>
}
```

Campo	Tipo	Descrizione
category	string	categoria alla quale appartiene il prodotto.
product	string	nome del prodotto (univoco per categoria).
price	float	prezzo di partenza dell'asta.

1.5.2 Ricerca di un prodotto

Attraverso questo comando è possibile ricercare un prodotto all'interno di una categoria.

```
{
  "token": <token>,
  "msg_id": 42,
  "category": <category name>
  "product": <product name>
}
```

Campo	Tipo	Descrizione
category	string	Categoria alla quale appartiene il prodotto.
product	strin	Espressione da ricercare come nome del prodotto.

1.5.3 Lista tutti i prodotti

```
{
  "token": <token>,
  "msg_id": 43,
  "category": <category name>
}
```

Campo	Tipo	Descrizione
category	string	categoria della quale si vuole la lista di prodotti.

1.5.4 Messaggio di risposta

La risposta del server eredita il formato base, introducendo un nuovo codice di risposta:

Campo	Tipo	Descrizione
response	int	'4' categoria non esistente

1.6 Asta

Una volta che i prodotti sono stati inseriti i vari utenti posso partecipare ad un asta offrendo un prezzo per l'oggeto.
L'asta termina quando si raggiunge il tempo limite, o quando il venditore decide di chiduerla. In entrambi i casi i partecipanti verranno notificati E' anche possibile ritirare un offerta, ma solo nel caso sia l'offerta più alta.

1.6.1 Offerta

L'utente può effettuare un offerta per un prodotto già inserito.
L'offerta sarà valida solamente se non è doppia (l'offerta più alta al momento non è dell'utente) al fine di evitare che un utente possa far salire arbitrariamente il prezzo di un prodotto.
Nel caso l'offerta sia inferiore all'ultima offerta registrata allora riceveremo un errore.

```
{
  'token': s<token>,
  'msg_id': 51,
  'category': <category name>,
  'product': <product name>
  'price': <price>
}
```

Campo	Tipo	Descrizione
category	string	nome della categoria
product	string	nome del prodotto
price	float	valore dell'offerta

1.6.2 Ritiro di un offerta

E' possibile ritirare un offerta fatta ad un'asta, ma solo nel caso sia ancora l'offerta più alta.

```
{
  'token': s<token>,
  'msg_id': 52,
  'category': <category name>,
  'product': <product name>
}
```

Campo	Tipo	Descrizione
category	string	nome della categoria
product	string	nome del prodotto

1.6.3 Chiusura di un'asta

```
{
  'token': s<token>,
  'msg_id': 53,
  'category': <category name>,
  'product': <product name>
}
```

Campo	Tipo	Descrizione
category	string	nome della categoria
product	string	nome del prodotto

1.6.4 Messaggio di risposta

La risposta del server eredita il formato base, introducendo nuovi codici di risposta:

Campo	Tipo	Descrizione
response	int	'4' categoria non valida
response	int	'5' asta non valida
response	int	'6' offerta non valida

2 Implementazione

Questo sisistema è stato implementato in Python 2.7, si divide in Client e Server, i quali sono composti da varie classi:

2.1 Server

Il server è stato implementato utilizzando più Thread. Il thread principale si occupa solamente di accettare le connessioni dagli utenti. Per ogni connessione viene fatto partire un thread ”ClientManager” che si occupa della comunicazione con il client.

2.1.1 IbaiServer

Questa è la classe principale del Server. Ci sono le funzioni di inizializzazione del server e del database. Contiene anche il loop principale all'interno del quale accetta le connessioni dai client e le passa alla classe ClientManager.

2.1.2 ClientManager

Questa è la classe che esegue la maggior parte delle operazioni relative ai client. Eredita la classe thread e definisce un metodo ”run” che viene eseguito all'avvio del thread.
Questo metodo è un loop di iterazione che riceve i comandi dal client. Una ricevuto un comando viene passato alla funzione “read_command” che lo interpreta e esegue la funzione appropriata. Le altre funzioni sono funzioni specifiche per ogni comando definito nel protocollo: sell, buy, bid, etc.

Sincronizzazione I vari thread del server accedono in maniera concorrente alla memoria condivisa del thread principale. Qui sono memorizzati gli utenti, le aste e le categorie. Per evitare problemi di accessi contemporanei alla memoria, tutte le funzioni che vi accedono sono decorate con una funzione detta “Synchronizer” che impedisce a due thread di eseguire un operazione sulla memoria contemporaneamente.

2.1.3 model

Auction

Category

User

Exceptions

2.2 Client

2.2.1 IbaiClient

2.3 Librerie utilizzate

3 Tests