JCP: JAVA4 Chat Protocol

Thursday, 27. April 2017

Johannes Posch

Abstract

In this document the protocol to communicate with Login-Server and Chat-Server is defined.

Contents

1 General **3**

2 Login-Server **4**

2.1 Command-LOGIN . . . 4

2.1.1 Parameters . . . 4

2.1.2 Response . . . 4

2.2 Command-LOGOUT . . . 5

2.2.1 Parameters . . . 5

2.2.2 Response . . . 5

2.3 Command-REGISTER . . . 5

2.3.1 Parameters . . . 5

2.3.2 Response . . . 5

3 Chat-Server **6**

3.1 Command-GET_USERS . . . 6

3.1.1 Parameters . . . 6

3.1.2 Response . . . 6

3.2 Command-GET_LOGGED_IN . . . 7

3.2.1 Parameters . . . 7

3.2.2 Response . . . 7

3.3 Command-GET_USER_IP . . . 7

3.3.1 Parameters . . . 7

3.3.2 Response . . . 7

4 Error-codes **8**

1 General

In this document:

- the sign **<...>** is defined as any value
- the sign **>...<** is defined as zero, one or more other array values

2 Login-Server

This server provides a fully functional interface to register users and manage their sessions. Talking to this server is done by sending JSON objects. Also the server responds with JSON objects.

!important The JSON object must be sent as a one line string ***!important***

The JSON object's contains different attributes and one is necessary to talk to the server. The "cmd" attribute is used by the server to authenticate the following actions.

At this point the protocol knows 3 different commands:

1. LOGIN
2. LOGOUT
3. REGISTER

The basic JSON-object then has at least the "cmd" attribute.

```
{"cmd":"<...>", ...}
```

Every command has it's own set of attributes, which are defined below. The server responds to every request with a description of the command which was called in first place. Afterwards it returns elements depending on the request. A basic response would be:

```
{"response":"<...>", ...}
```

2.1 Command-LOGIN

This command is used as described to login to the server. An existing user is needed to perform the action successful. The server does a lookup for the given parameters in the background. On a successful query the server returns a new valid token. If there was a failure no token will be sent but a error-code.
The Login command's JSON as an example would be:

```
{"cmd":"LOGIN", "user":"<...>", "pw":"<...>"}
```

2.1.1 Parameters

- "user" [string]: the user-name for login
- "pw" [string]:the password for the user

2.1.2 Response

The response to this command contains an error-code, a success flag and a new token. An example of the response-object would be:

```
{"response":"LOGIN", "success":(true | false), "error":<...>, "token":"<...>"}
```

2.2 Command-LOGOUT

This command is used to logout the user with a valid token from the server. The server deletes the given token from it's database. On success the server returns a success-flag set. On error an error-code will be returned.
The Logout command's JSON as an example would be:

```
{"cmd":"LOGOUT", "token":"<...>"}
```

2.2.1 Parameters

- "token" [string]: This token represents the user which should be invalidated

2.2.2 Response

The response to this command contains an error-code and a success flag. An example of the response-object would be:

```
{"response":"LOGOUT", "success":(true | false), "error":<...>}
```

2.3 Command-REGISTER

The command is used to add a completely new user to the system. The username, needs to be unique. The server creates entries to the data given in it's database. On success the server returns a success-flag. On error an error-code will be returned.
The Register command's JSON as an example would be:

```
{"cmd":"REGISTER", "firstname":"<...>", "secondname":"<...>", "user":"<...>",
"pw":"<...>"}
```

2.3.1 Parameters

- "firstname" [string]: The firstname of the new user
- "secondname" [string]: The secondname of the new user
- "user" [string]: The user-name for the new user
- "pw" [string]: The password for the new user

2.3.2 Response

The response to this command contains an error-code and a success flag. An example of the response-object would be:

```
{"response":"REGISTER", "success":(true | false), "error":<...>}
```

3 Chat-Server

This server is connected to the Login-Server and handles chat traffic through the whole system. Talking to this server is done by sending JSON objects. Also the server responds with JSON objects.

!important The JSON object must be sent as a one line string ***!important***

The JSON object's contains different attributes and one is necessary to talk to the server. The "cmd" attribute is used by the server to authenticate the following actions.

At this point the protocol knows 3 different commands:

1. GET_USERS
2. GET_LOGGED_IN
3. GET_USER_IP

The basic JSON-object then has at least the "cmd" attribute.

```
{"cmd":"<...>", ...}
```

Every command has it's own set of attributes, which are defined below. The server responds to every request with a description of the command which was called in first place. Afterwards it returns elements depending on the request. A basic response would be:

```
{"response":"<...>", ...}
```

3.1 Command-GET_USERS

This command is used to get a full list of all users from the server. To access the server a valid token has to be passed to the system. On success a success-flag is set. On error an error-code is returned. The Login command's JSON as an example would be:

```
{"cmd":"GET_USERS", "token":"<...>"}
```

3.1.1 Parameters

- "token" [string]: a valid token to access the server

3.1.2 Response

The response to this command contains an error-code, a success flag and a list of available user-names. An example of the response-object would be:

```
{"response":"GET_USERS", "success":(true | false), "error":<...>,
"users":[{"name":"<...>"}, >...<}]}
```

3.2 Command-GET_LOGGED_IN

This command returns all users (their user-name) which have a valid token in the system. To access the server a valid token has to be passed to the system. On success a success-flag is set. On error an error-code is returned.
The Login command's JSON as an example would be:

```
{"cmd":"GET_LOGGED_IN", "token":"<...>"}
```

3.2.1 Parameters

- "token" [string]: a valid token to access the server

3.2.2 Response

The response to this command contains an error-code, a success flag and a list of available user-names. An example of the response-object would be:

```
{"response":"GET_LOGGED_IN", "success":(true | false), "error":<...>,
"users":[{"name":"<...>"}, >...<}]}
```

3.3 Command-GET_USER_IP

This command returns a list of IP addresses of a logged in user (a user with a valid token). To access the server a valid token has to be passed to the system. On success a success-flag is set. On error an error-code is returned.
The Login command's JSON as an example would be:

```
{"cmd":"GET_USER_IP", "token":"<...>", "user":"<...>"}
```

3.3.1 Parameters

- "token" [string]: a valid token to access the server
- "user" [string] the user-name of the requested user

3.3.2 Response

The response to this command contains an error-code, a success flag and a list of available user-names. An example of the response-object would be:

```
{"response":"GET_USER_IP", "success":(true | false), "error":<...>,
"user_ip":[{"ip":"<...>"}, >...<}]}
```

4 Error-codes

The response can also contain an error-code. This code is set if an error occurred on the server. The codes, which are returned have different meanings:

- 0 [int]: no error occurred, everything worked normal
- 1 [int]: the given user-name is already in use
- 2 [int]: the password is incorrect
- 3 [int]: the given token is invalid
- 4 [int]: the user is not defined
- 5 [int]: the requested user does not exist
- 6 [int]: the requested user is not logged in
- 20 [int]: a database failure occurred
- 21 [int]: a server failure occured
- 50 [int]: invalid JSON object
- 51 [int]: invalid parameters in JSON object
- 52 [int]: a parameter value was not set
- 53 [int]: unknown command